POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Physics-Informed Neural Networks for Shallow Water Equations

Laurea Magistrale in Aeronautical Engineering - Ingegneria Aeronautica

Author: Riccardo Anelli

Advisor: Prof. Edie Miglio

Academic year: 2021-2022

---

1. Introduction

The massive increase of data and computing resources available made possible, over the last 15 years, a significant growth in the field of machine learning—particularly, in deep learning. The term *deep learning* generally refers to Neural Network (NN) methods. These methods are able to extract patterns and models automatically from large volumes of data, and are generally agnostic to the underlying scientific principles driving the variables. However, in scientific problems the variables can interact in complex nonstationary and nonlinear ways, and the available dataset is often limited: this makes considerably difficult the challenge of achieving good performances with data-hungry methods.

Recently, a class of methods that incorporate Partial Differential Equations (PDEs) into a NN emerged, these methods are commonly referred to as Physics-Informed Neural Networks (PINNs) [6]. This technique can automatically extract patterns from data *while* taking advantage of the theoretical knowledge accumulated in scientific theories. The fundamental step of this approach is embedding a PDE into the loss of the NN using automatic differentiation: by constraining the NN to minimize the PDE residual, the space of admissible solutions the NN can identify is shrunk to the physical ones.

1.1. Objective

The objective of this work is to investigate the possibility of using the PINNs to approximate the Shallow Water Equations, a system of hyperbolic PDEs that simulates free-surface flow problems.

Specifically, parametric cases are intentionally built upon selected benchmark problems, for the purpose of testing the degree to which the PINN is a convenient tool to be used when dealing with *many-query* problems. The PINN seems to be a promising tool to deal with this class of problems for one fundamental reason: following one training session, the PINN is able to instantly provide simulations for every requested parameter value. Consequently, the computational cost of the complete parameter space analysis is, basically, the one of a single training session.

2. Mathematical model

While the Navier–Stokes equations are a comprehensive model that can describe the motion of a three-dimensional *real* fluid, the computational cost of a three-dimensional model simulation is very high. It makes sense to reduce the model for calculations with simpler flow conditions. The depth-averaged two-dimensional flow equations, also called Shallow Water Equations (SWE), provide a suitable approximation

to model free-surface flow problems. Within the ambit of this work the equations are written in conservative form, in the Cartesian coordinate system, following the notation illustrated in Fig. 1.

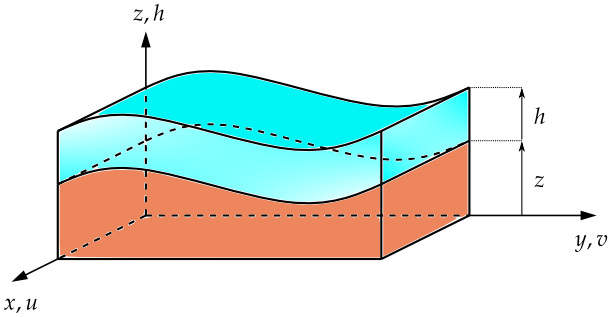


Figure 1: 2D SWE. Notation.

The expression of the *inviscid* 2D SWE, holding as unknowns the water height $h(x, y, t)$ and the two horizontal components of the depth-averaged velocity $u(x, y, t)$ and $v(x, y, t)$, is given in Eq 1.

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{E}}{\partial x} + \frac{\partial \mathbf{G}}{\partial y} = \mathbf{S} \tag{1}$$

The detailed expression of $\mathbf{U}$, $\mathbf{E}$, $\mathbf{G}$, $\mathbf{S}$ are:

$$\mathbf{U} = \begin{bmatrix} h \\ hu \\ hv \end{bmatrix},$$

$$\mathbf{E} = \begin{bmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \\ huv \end{bmatrix}, \quad \mathbf{G} = \begin{bmatrix} hv \\ huv \\ hv^2 + \frac{1}{2}gh^2 \end{bmatrix},$$

$$\mathbf{S} = \begin{bmatrix} 0 \\ -gh(\frac{\partial z}{\partial x} + \frac{1}{C^2}\frac{u|\mathbf{u}|}{h}) \\ -gh(\frac{\partial z}{\partial y} + \frac{1}{C^2}\frac{v|\mathbf{u}|}{h}) \end{bmatrix}$$

where:

- the coordinate z defines the vertical direction, to which the free surface elevation is associated;
- g is the standard gravitational acceleration;
- $\mathbf{u} = [u, v]^T$ is the velocity vector;
- C is the Chézy's friction coefficient.

Additional force terms can be included in the term $\mathbf{S}$, depending on the chosen application. In order to properly pose this PDE problem, Boundary Conditions (BC) and Initial Conditions (IC) constraints are needed. Moreover, the problems examined throughout this work are often defined in big spatial domains, develop within large time scales, or involve infinitesimal solutions. Since the NN best deals with both domains and solutions belonging to the $\mathcal{O}(1)$ scale, the effective strategy that can get every single problem to fit with the solver is the *scaling*.

3. Neural Networks

Mathematically, the NN is a compositional function. The simplest NN is the Feedforward Neural Network (FNN), which applies linear and nonlinear transformations to the inputs recursively. Although many different types of NNs have been developed in the past decades, the FNN is suitable for most PDE problems [4] and it will be the only NN variety considered in this work. A typical FNN architecture is illustrated in Fig. 2.

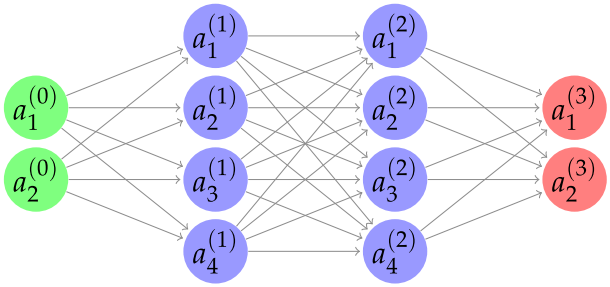


Figure 2: Example of FNN.

The constituent parts that make up the FNN's architecture and terminology are:

- the neurons;
- the layers;
- the neurons per layer;
- the activation functions;
- the weights;
- the biases;
- the loss function.

Moreover, the fundamental tools required by this technology are:

- the Automatic Differentiation (AD), required to compute the gradients of the loss function with respect to the NN's weight and biases, and, for the PINN, to compute the derivatives of the NN outputs with respect to its inputs;
- the stochastic gradient descent.

4. Physics-Informed Neural Networks

The PINNs are a specific type of NNs trained to approximate the solution to any given law of physics described, in general, by a PDE or by a system of PDEs. This kind of approach allows to use the PDEs in *strong form* directly.

4.1. Partial differential equations

Let $u(\mathbf{x})$ be an unknown function defined in the spatio-temporal domain $\Omega \subset \mathbb{R}^d$, let $\boldsymbol{\lambda}$ be the set of data on which the PDE depends. For the sake of simplicity, a second-order differential equation is considered. In this case, the generic PDE is expressed as:

$$\mathcal{P}(u, \boldsymbol{\lambda}) = f\Bigg(\mathbf{x}, \frac{\partial u}{\partial x_1}, \dots, \frac{\partial u}{\partial x_d}, \frac{\partial^2 u}{\partial x_1 \partial x_1}, \dots, \frac{\partial^2 u}{\partial x_1 \partial x_d}, \dots, \boldsymbol{\lambda}\Bigg) = 0$$

where $\mathbf{x} = (x_1, \dots, x_d)$. Any of the variables x_i, for $i = 1, \dots, d$, could represent the temporal variable, consequently, the IC is dealt with as a special case of Dirichlet data on the spatio-temporal border $t = 0$. Suitable BC can be symbolized in the following form:

$$\mathcal{B}(u, \mathbf{x}) = 0 \quad \text{on } \partial\Omega$$

where $\mathcal{B}(u, \mathbf{x})$ could be Dirichlet, Neumann, Robin, even time-depending BC.

4.2. The PINN algorithm

The PINN algorithm is simple, and it can be described in four steps:

Step 1: a NN $\hat{u}(\mathbf{x}, \boldsymbol{\theta})$ is built as a surrogate of the unknown solution $u(\mathbf{x})$, where $\mathbf{x} \in \mathbb{R}^d$.

$\boldsymbol{\theta}$ is the set of all the NN's parameters, so that: $\boldsymbol{\theta} = \{\mathbf{W}^\ell, \mathbf{b}^\ell\}_{1 \le \ell \le L}$.

$\mathbf{W}^\ell \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$ and $\mathbf{b}^\ell \in \mathbb{R}^{N_\ell}$ are the weight matrix and the bias vector in the ℓ-th layer, respectively, and L is the number of layers of the NN.

The NN's input neurons number matches d, while the NN's output neurons number matches the dimension of u.

Step 2: two sets of scattered training points $\mathcal{T}_f \subset \Omega$ and $\mathcal{T}_b \subset \partial\Omega$ are defined: these are the locations in the computational domain where $\hat{u}$ will be trained, i.e., $\hat{u}$ is constrained to satisfy the PDE in these points.

The whole training points set is defined as $\mathcal{T} = \{\mathcal{T}_f, \mathcal{T}_b\}$.

Step 3: a suitable loss function is defined to make an estimation of the discrepancy between $\hat{u}$ and the constraints, the PDE and the BC/IC, whose residuals are respectively symbolized as $\mathcal{P}(\hat{u}, \boldsymbol{\lambda})$ and $\mathcal{B}(\hat{u}, \mathbf{x})$:

$$\mathcal{L}(\boldsymbol{\theta}, \mathcal{T}) = w_f \mathcal{L}_f(\boldsymbol{\theta}, \mathcal{T}_f) + w_b \mathcal{L}_b(\boldsymbol{\theta}, \mathcal{T}_b)$$

where w_f and w_b are the weights, and:

$$\mathcal{L}_f(\boldsymbol{\theta}, \mathcal{T}_f) = \frac{1}{|\mathcal{T}_f|} \sum_{x \in \mathcal{T}_f} \|\mathcal{P}(\hat{u}, \boldsymbol{\lambda})\|_2^2$$

$$\mathcal{L}_b(\boldsymbol{\theta}, \mathcal{T}_b) = \frac{1}{|\mathcal{T}_b|} \sum_{x \in \mathcal{T}_b} \|\mathcal{B}(\hat{u}, \mathbf{x})\|_2^2$$

All the derivatives within both the residuals $\mathcal{P}(\hat{u}, \boldsymbol{\lambda})$ and $\mathcal{B}(\hat{u}, \mathbf{x})$ are effectively handled via AD.

Step 4: the loss function $\mathcal{L}(\boldsymbol{\theta}, \mathcal{T})$ is minimized through the *training* of the neural network $\hat{u}$. This procedure will constantly update the parameters set $\boldsymbol{\theta}$ up to an optimal composition $\boldsymbol{\theta}^*$ that will get $\hat{u}$ to behave the desired way.

A diagram of the whole procedure—for the case of the 1D heat equation supplied with mixed BCs—is shown in Fig. 3.

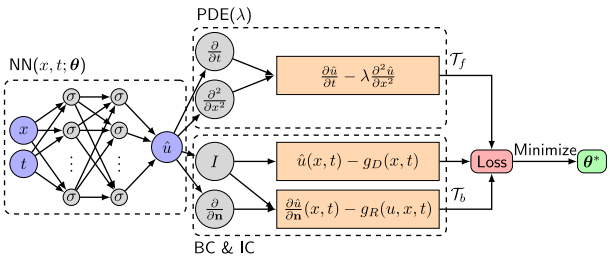


Figure 3: PINN algorithm. Application to the 1D heat equation. (from Ref. [4]).

4.3. Error analysis

The intrinsic limits of the NN approach are due to the fact that:

- The NN have inevitably limited size. Let $\mathcal{F}$ be the family of all the functions representable by a given finite-size NN; the *best* function representable by the NN, i.e., the closest to u, is defined as: $u_{\mathcal{F}} = \text{argmin}_{f\in\mathcal{F}}\|f - u\|$.

  The **approximation error** $\mathcal{E}_{\text{app}}$ is defined as: $\|u_{\mathcal{F}} - u\|$.

- The NN are trained on a finite set of training points. The NN's representable function if the loss is at the global minimum is defined as: $u_{\mathcal{T}} = \text{argmin}_{f\in\mathcal{F}}\mathcal{L}(f, \mathcal{T})$.

  The **generalization error** $\mathcal{E}_{\text{gen}}$, determined both by the density of the training points and by the NN's expressiveness, is defined as: $\|u_{\mathcal{T}} - u_{\mathcal{F}}\|$.

- Minimizing the non-convex, highly nonlinear, loss function can be computationally unmanageable, hence, the computation of the global minimum of $\mathcal{L}$ is a task very unlikely to be performed. Let $\tilde{u}_{\mathcal{T}}$ be the actual function represented by the NN as a result of the training.

  The **optimization error** $\mathcal{E}_{\text{opt}}$ is defined as: $\|\tilde{u}_{\mathcal{T}} - u_{\mathcal{T}}\|$.

Hence, the total error $\mathcal{E}$ is defined as:

$$\mathcal{E} \overset{\text{def}}{=\!=} \|\tilde{u}_{\mathcal{T}} - u\| \leq \mathcal{E}_{\text{app}} + \mathcal{E}_{\text{gen}} + \mathcal{E}_{\text{opt}}$$

The illustration of this error decomposition is shown in Fig. 4.

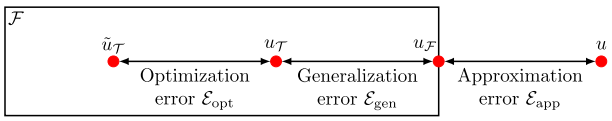


Figure 4: Error analysis. Decomposition of the total error (from Ref. [4]).

5. Results

The PINN tool is applied to some preliminary elliptic, parabolic and hyperbolic models, then, to ten case tests built on the SWE model. For the sake of brevity, two problems are picked and shown in this document. All the tests are conducted on a NVIDIA® T4 GPU, a cloud computing resource freely accessible in Google Colaboratory.

5.1. Dam break on a dry domain with friction, *parametric*

This case is built upon the Dressler's dam break with friction presented in [2]. The problem is defined in the domain $\Omega \times (0, T)$, with:

$$\Omega = (x_{\min}, x_{\max}) \times (y_{\min}, y_{\max})$$

and $x \in (0, 2000)$, $y \in (0, 1)$, $t \in (0, 40)$. The additional variable y accounts for the parameter space. The dam break is instantaneous and the bottom is flat.

The initial conditions are:

$$h(x, y, 0) = \begin{cases} 6 \text{ m} & x \leq 1000 \text{ m} \\ 0 \text{ m} & x > 1000 \text{ m} \end{cases}$$

and

$$u = 0 \text{ m/s}$$

No boundary condition is required to be set. The additional parameter space is integrated in the mathematical formulation by way of a linear transformation between the variable y and the Chézy's friction coefficient, C, actually entering the PDE:

$$C = 100y + 20$$

This technique is used with the purpose of keeping the parameter space dimension unitary, while obtaining a Chézy's friction coefficient ranging extensively from the values 20 and 120.

The SWE are implemented holding as unknowns h and u, plus, two supplementary inequalities are included in the system for the purpose of making the learning faster. The complete system is given in Eq. (2).

$$\begin{cases} \dfrac{\partial h}{\partial t} + \dfrac{\partial (hu)}{\partial x} = 0 \\ \dfrac{\partial (hu)}{\partial t} + \dfrac{\partial (hu^2 + \frac{1}{2} g h^2)}{\partial x} = -g \dfrac{u|\mathbf{u}|}{C^2} \\ h \geq 0 \\ u \geq 0 \end{cases} \tag{2}$$

Model setup. Considering the large spatio-temporal domain, the model *has* to be scaled. Specifically, the reference values for the scaling of the space (**L**), time (**T**), water height (**H**) and velocity (**U**) are set as:

- $\mathbf{L} = 2000$
- $\mathbf{T} = 40$
- $\mathbf{H} = 6$
- $\mathbf{U} = u_{\max}^{120}$

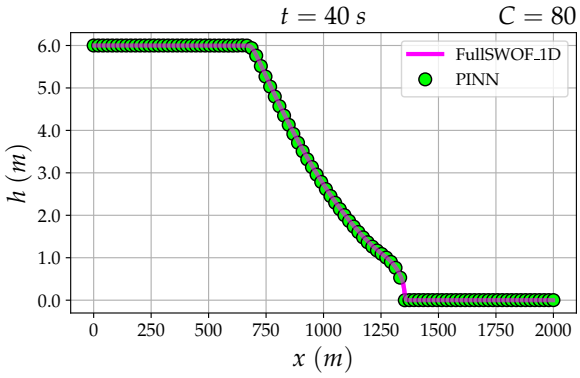
where $u_{\max}^{120}$ is the maximum value of the numerical solution for the water velocity, considering $C = 120$, computed by FullSWOF_1D [1].

Hyperparameters. For the training is used an Adam optimizer and a fixed learning rate of 10^{-4}, the hyperbolic tangent is used as activation function, and the network architecture is set to 4 hidden layers with 60 neurons per layer. The training points in the domain and for the IC are 2×10^4 and 10^3, respectively. No training point is set on the boundary. The number of iterations is 5×10^5.

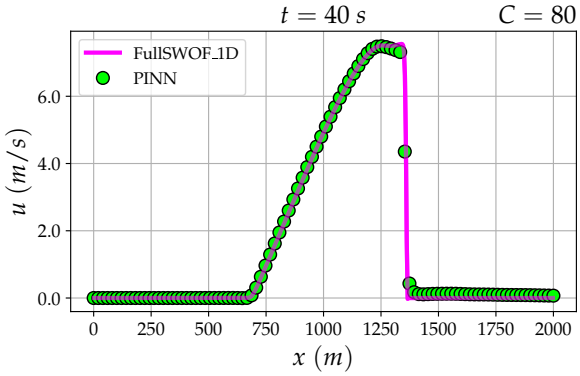
The loss function is made up of six terms: ($\mathcal{L}^h_{PDE}$, $\mathcal{L}^{hu}_{PDE}$, $\mathcal{L}^h_{ineq}$, $\mathcal{L}^u_{ineq}$ $\mathcal{L}^h_{IC}$, $\mathcal{L}^u_{IC}$), respectively generated by the residuals of the mass conservation and of the momentum balance equations, the inequalities and the ICs for the variables h and u. Every single term enters the loss function with an associated weight that proved to work fine after a trial-and-error effort:

$$\mathcal{L} = \mathbf{10^2}\mathcal{L}^h_{PDE} + \mathbf{10^2}\mathcal{L}^{hu}_{PDE} + \mathbf{10^5}\mathcal{L}^h_{ineq} + \mathbf{10^5}\mathcal{L}^u_{ineq} + \mathbf{10^5}\mathcal{L}^h_{IC} + \mathbf{10^5}\mathcal{L}^u_{IC}$$

The results of the learning of the variables h and u, for one selected value of the parameter C, are shown in Fig. 5. Examining the plots, the PINN predictions are a little bit smoother than the FullSWOF_1D approximation. However, the features of this case study—particularly, the position of the shock—appear to be identified almost perfectly.



(a)



(b)

Figure 5: Dam break on a dry domain with friction, parametric. PINN solutions for **(a)** the water height, and **(b)** the velocity. Comparison with the FullSWOF_1D results.

5.2. Circular dam break, *parametric*

This case is built upon the circular dam break presented in [3]. The problem is defined in the domain $\Omega \times (0, T)$, with:

$$\Omega = (x_{\min}, x_{\max}) \times (y_{\min}, y_{\max}) \times (z_{\min}, z_{\max})$$

and $x \in (-25, 25)$, $y \in (-25, 25)$, $z \in (1, 20)$, $t \in (0, 1)$. The additional variable z accounts for the parameter space. Still, the dam break is instantaneous and the bottom is flat.
The initial conditions are:

$$h(x, y, z, 0) = \begin{cases} 10 \text{ m} & |x| \leq \sqrt{100 - y^2}, \\ & |y| \leq \sqrt{100 - x^2} \\ 1 \text{ m} & \text{otherwise} \end{cases}$$

and

$$u(x, y, z, 0) = 0 \text{ m/s}$$
$$v(x, y, z, 0) = 0 \text{ m/s}$$

while the boundary conditions are:

$$u(x_{\min}, y, z, t) = 0 \text{ m/s}$$
$$u(x_{\max}, y, z, t) = 0 \text{ m/s}$$

and

$$v(x, y_{\min}, z, t) = 0 \text{ m/s}$$
$$v(x, y_{\max}, z, t) = 0 \text{ m/s}$$

The SWE are implemented holding as unknowns h, u and v. The gravitational acceleration, g, enters the PDE as a wide-ranging parameter belonging to the third spatial axis of the problem, i.e., z. The complete system is given in Eq. (3).

$$\begin{cases} \dfrac{\partial h}{\partial t} + \dfrac{\partial (hu)}{\partial x} + \dfrac{\partial (hv)}{\partial y} = 0 \\ \dfrac{\partial (hu)}{\partial t} + \dfrac{\partial (hu^2 + \frac{1}{2} z h^2)}{\partial x} + \dfrac{\partial (huv)}{\partial y} = 0 \\ \dfrac{\partial (hv)}{\partial t} + \dfrac{\partial (huv)}{\partial x} + \dfrac{\partial (hv^2 + \frac{1}{2} z h^2)}{\partial y} = 0 \end{cases} \quad (3)$$

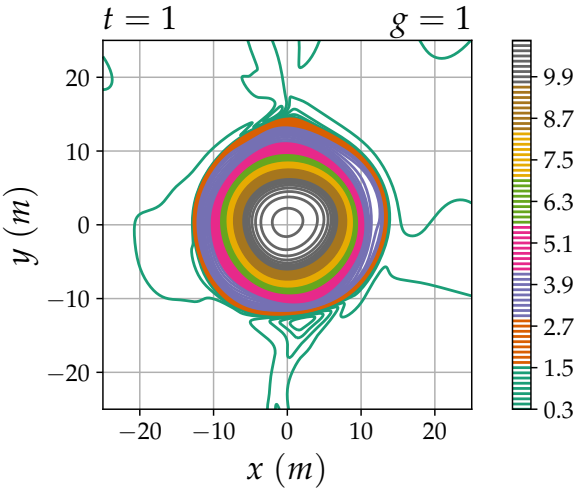
Remark. In spite of the large domain, the model is not scaled.

Hyperparameters. For the training are used, sequentially, Adam and L-BFGS optimizers. The learning rate is fixed at 10^{-3}, the hyperbolic tangent is used as activation function, and the network architecture is set to 4 hidden layers with 30 neurons per layer. The training points in the domain, on the boundary and for the IC are 1.2×10^4, 5×10^3 and 1.2×10^4, respectively. The number of iterations is 3×10^5 with the Adam optimizer and 2×10^5 with the L-BFGS.

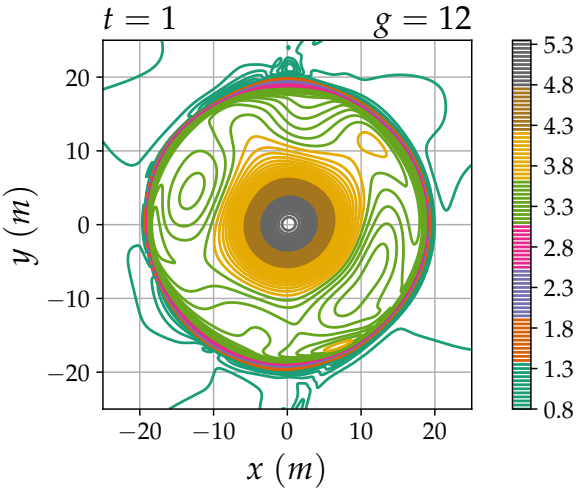
The loss function is made up of ten terms: ($\mathcal{L}^h_{PDE}$, $\mathcal{L}^{hu}_{PDE}$, $\mathcal{L}^{hv}_{PDE}$, $\mathcal{L}^h_{IC}$, $\mathcal{L}^u_{IC}$, $\mathcal{L}^v_{IC}$, $\mathcal{L}^u_{BC_{x_{\min}}}$, $\mathcal{L}^u_{BC_{x_{\max}}}$, $\mathcal{L}^v_{BC_{y_{\min}}}$, $\mathcal{L}^v_{BC_{y_{\max}}}$), respectively generated by the residuals of the mass conservation and of two momentum balance equations, the three ICs and the four BCs. Every term enters the loss function with an associated weight set to 1:

$$\begin{aligned} \mathcal{L} = & \mathbf{1}\mathcal{L}^h_{PDE} + \mathbf{1}\mathcal{L}^{hu}_{PDE} + \mathbf{1}\mathcal{L}^{hv}_{PDE} \\ & + \mathbf{1}\mathcal{L}^h_{IC} + \mathbf{1}\mathcal{L}^u_{IC} + \mathbf{1}\mathcal{L}^v_{IC} \\ & + \mathbf{1}\mathcal{L}^u_{BC_{x_{\min}}} + \mathbf{1}\mathcal{L}^u_{BC_{x_{\max}}} \\ & + \mathbf{1}\mathcal{L}^v_{BC_{y_{\min}}} + \mathbf{1}\mathcal{L}^v_{BC_{y_{\max}}} \end{aligned}$$

The results of the learning of the variable h, for two selected values of the parameter g, are shown in Fig. 6. Examining the plots, the features of this case study seem to be replicated pretty well. Specifically, the PINN predictions preserve the symmetry of the problem.



(a)



(b)

Figure 6: Circular dam break, parametric. PINN solutions for the water height.

6. Conclusions

The extensive PINN application showed that the *accuracy* of the numerical solutions produced is solid: the integral errors, as well as the qualitative inspection of the results, gave evidence of an extremely accurate tool, regardless of the the presence of viscosity or shock waves.

Dealing with *parametric* problems, the PINNs shown their suitability: once that the network was trained over the needed parameter space, every single recall of the model for the requested parameter value was immediate. This happened in light of the fact that these recalls were only evaluations of a trained network.

PINNs proved to work well when the *dimensions* of the problems were increased. There is no mesh to be built. One more dimension for the problem domain translates into one more input-layer neuron, one more dimension for the problem solution translates into one more output-layer neuron.

Some factors that could be counted against the use of PINNs have been experienced. The computational cost of the PINN training is considerably high: the average time required for the training of every single model examined has been of about *four* hours. Moreover, the choices of the effective hyperparameters' values, and of the proper scaling of the model, is currently done empirically through a tough trial-and-error task.

Due to these challenges, PINN is currently not a believable alternative to traditional methods like Finite Element Method (FEM) or Finite Volume Method (FVM). But those methods take advantage of a 50-years-long process of development, so, daring a comparison today is not even fair. Putting things in perspective, since research in the neural network field is very active, great improvements are expected.

Concerning the design of the most effective neural network architecture for the problem at hand, there's still experience to be accumulated before it ceases to be done empirically by the user. Finally, in order to reduce the computational cost of the training, the clustered residual points distribution [5]—a smaller set of points placed were they matter the most, without *a priori* knowledge of the solution—even for the parametric cases, would definitely help.

References

[1] O. Delestre, F. Darboux, F. James, C. Lucas, C. Laguerre, S. Cordier, *FullSWOF: Full Shallow-Water equations for Overland Flow*, Journal of Open Source Software, 2(20):448, 2017.

[2] O. Delestre, C. Lucas, P.A. Ksinant, F. Darboux, C. Laguerre, T.N.T. Vo, F. James, S. Cordier, *SWASHES: a compilation of Shallow Water Analytic Solutions for Hydraulic and Environmental Studies*, International Journal for Numerical Methods in Fluids, 72(3):269–300, 2013.

[3] G.F. Lin, J.S. Lai, W.D. Guo, *Finite-volume component-wise TVD schemes for 2D shallow water equations*, Advances in Water Resources, 26(8):861–873, 2003.

[4] L. Lu, X. Meng, Z. Mao, G.E. Karniadakis, *DeepXDE: A deep learning library for solving differential equations*, arXiv:1907.04502v2, 2020.

[5] Z. Mao, A.D. Jagtap, G.E. Karniadakis, *Physics-informed neural networks for high-speed flows*, Computer Methods in Applied Mechanics and Engineering, 360:112789, 2020.

[6] M. Raissi, P. Perdikaris, G.E. Karniadakis, *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, 2019.