Programming Practical: Physics-Informed Neural Networks for simulation in hydraulics

H. Boulenc, J. Monnier

March 2024

Contents

1 Neural Networks with Pytorch

Pytorch is a machine learning framework developed by Meta AI since 2016. It is free, open source, and the two main features it allows are :

- Automatic differentiation system via a computational graph.
- Tensor computing with strong acceleration via GPU.

To get a better introduction to the different objects and functionalities that this library offers, a notebook titled "Intro_to_Pytorch.ipynb" is provided. Please have a look and try to experiment a bit with the code to get a grasp of what is possible. The answers to most of your questions are probably inside this notebook, so take the time to understand it well !

2 Physical model

For the scope of this Programming Practical, a simplified 1D permanent hydraulics model called the backwater equation model is detailed below. The objective of this algorithm is to approximate the solution $x \mapsto h(x) \quad \forall x \in \Omega$ of the model defined after. The reference solution is obtained by integrating the backwater equation using a RK4 numerical scheme.

For the backwater equation, the following formulation is used:

$$h'(x) = -\frac{b'(x) + j(\boldsymbol{K_s}; h, x)}{1 - Fr^2(h(x))}, \quad j(\boldsymbol{K_s}; h, x) = \frac{q^2}{\boldsymbol{K_s}^2(x) h(x)^{10/3}}, \; Fr^2(h(x)) = \frac{q^2}{g h(x)^3} \quad \forall x \in \Omega \tag{1}$$

With $h(x)$, $b(x)$ and $Fr(x)$ respectively the water height, the bathymetry and the Froude number at location x. The following constants are also given: q for the flow rate per unit width in $m^3/s/m$, g for the gravity and h_0 for the upstream or downstream water height boundary condition depending on the flow regime, see Equation 2.

The solution h here depends on a spatially-distributed friction function parameter: the Strickler coefficient $x \mapsto \boldsymbol{K_s}(s) \quad \forall x \in \Omega$, with $\boldsymbol{K_s}(x)$ in $m^{1/3}.s^{-1}$. The objective of this PP is to solve the backwater equation for a given friction $\boldsymbol{K_s} \in \mathcal{K}$ using neural networks.

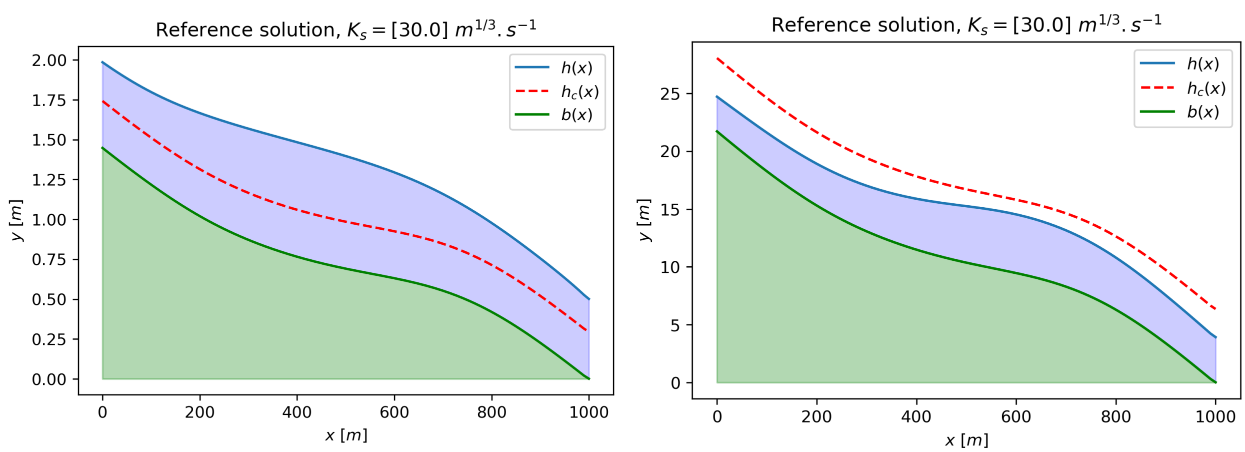

Figure 1: Regimes for backwater equation (left: subcritical, right: supercritical)

3 Physics-Informed Neural Networks

3.1 Physical residual

From the direct model introduced in Equation 1, the physical residual can be defined as:

$$r(\boldsymbol{K_s}; h)(x) = h'(x) + \frac{b'(x) + j(\boldsymbol{K_s}; h, x)}{1 - Fr^2(h(x))} \quad \forall x \in \Omega$$

Once the physical residual is defined, a grid of N_{col}^x colocation points $\mathcal{X}_{col} = \{x_{col}^{(i)}\}_{i=1,\dots,N_{col}^x}$ can be sampled in the domain Ω, regularly or not, to minimize the residual on its vertices, called colocation points. Let $\|\cdot\|$ be the norm associated to the euclidean inner product. To embed prior physical knowledge into the training of the neural network, the following physical residual loss function can be minimized:

$$J_{res}(\boldsymbol{K_s}; h) = \frac{1}{N_{col}^x} \|r(\boldsymbol{K_s}; h)\|^2_{\mathcal{X}_{col}}$$

By using Automatic Differentiation tools (AD), derivatives can be evaluated for a low computational cost and the J_{res} loss function can easily be evaluated at the colocation points. Since y will be approximated by the output of a neural network, it is also important to note that the activation functions $\{\sigma_i\}_{i=1,\dots,d}$ have to be chosen regular enough to be differentiable a sufficient amount of times.

Furthermore, a second loss function J_{BC} is introduced to ensure that the solution will satisfy the boundary condition given with the direct model defined by Equation 1.

$$J_{BC}(h(x_{BC})) = (h(x_{BC}) - h_{BC})^2$$

With $x_{BC} = sup(\Omega)$ in subcritical regime, $x_{BC} = inf(\Omega)$ in supercritical regime and h_{BC} given.

To respect both physical constraints and data discrepancy during the training of $\mathcal{N}_{\boldsymbol{\theta}}$, the following total loss function can be minimized :

$$J(\boldsymbol{K_s}; h) = \lambda_{res} J_{res}(\boldsymbol{K_s}; h) \; + \; \lambda_{BC} J_{BC}(h(x_{BC}))$$

Where $\lambda_{res}, \lambda_{BC} \in \mathbb{R}$ are the scalarization factors for the multi-objective optimization problem.

3.2 Neural Networks with physical constraints

Let's consider a Neural Network $\mathcal{N}_{\boldsymbol{\theta}}$ of the following form :

$$\mathcal{N}_{\boldsymbol{\theta}} : I \mapsto \tilde{h}_{\boldsymbol{\theta}}(I)$$

With I the inputs of the Neural Network. This PP will investigate two architectures for I:

- Non-parametric inputs: $I = x \in \mathcal{X}_{col}$,
- Parametric inputs: $I = (\boldsymbol{K_s}; x) \in \mathcal{X}_{col} \times \mathcal{K}_{col}$ with $\mathcal{K}_{col}$ a grid of N_{col}^k colocation points $\mathcal{K}_{col} = \{\boldsymbol{K_s}_{col}^{(i)}\}_{i=1,\dots,N_{col}^k}$ sampled regularly or not in $\mathcal{K}$.

The training of $\mathcal{N}_{\boldsymbol{\theta}}$ as a Physics-Informed Neural Networks then refers to the following optimization problem:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{argmin} \left(J(\boldsymbol{K_s}; \tilde{h}_{\boldsymbol{\theta}}) \right)$$

In addition to that, a pre-training consisting in setting the value of $h(x)$ to h_{BC} for all colocation points is used before the training to begin the minimization of J_{res} in a good neighborhood of the physical solution.

$$\boldsymbol{\theta}^{pre} = \underset{\boldsymbol{\theta}}{argmin} \left(\|\tilde{h}_{\boldsymbol{\theta}}(I) - h_{BC}\|_2^2 \right) \quad \forall I \in \mathcal{I}_{col}$$

With $\mathcal{I}_{col} = \mathcal{X}_{col}$ or $\mathcal{I}_{col} = \mathcal{X}_{col} \times \mathcal{K}_{col}$ depending on the inputs architecture choice.

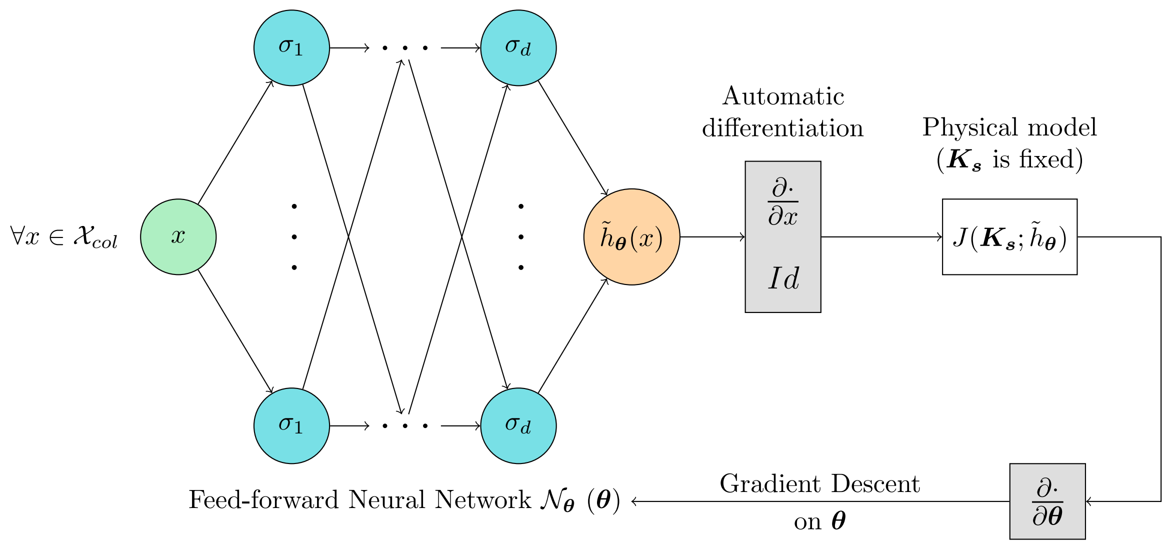

Figure 2: PINN for backwater direct modelling with non-parametric inputs

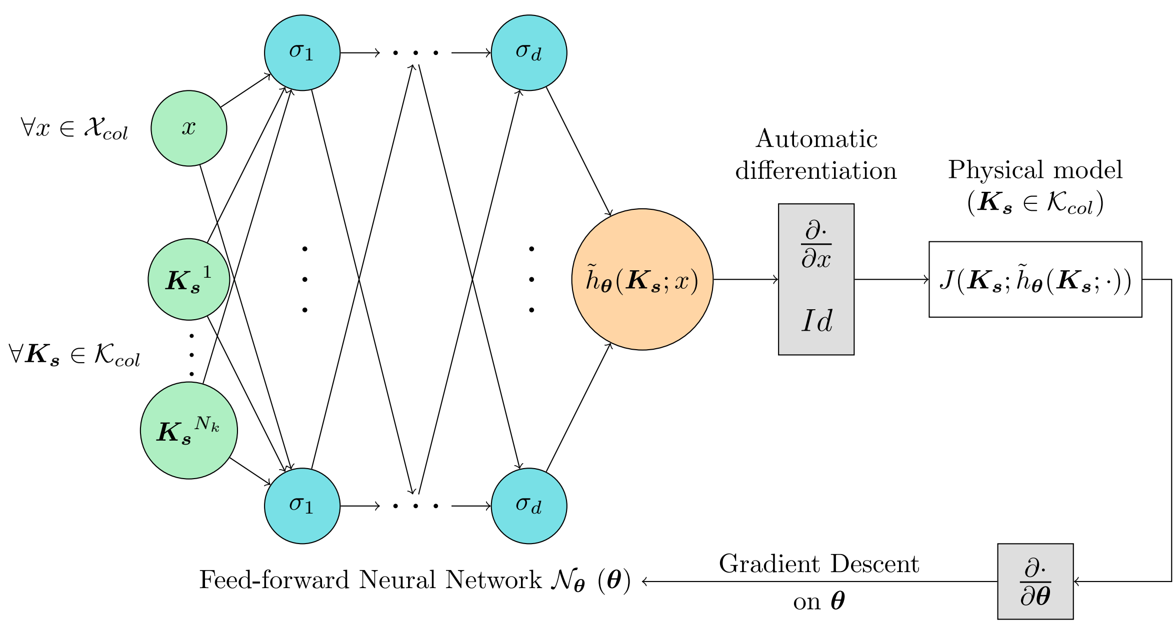

Figure 3: PINN for backwater direct modelling with parametric inputs

4 Your job

4.1 Theoretical study

First of all, you have to define the right loss functions J_{res} and J_{BC} such that when minimizing them, the solution of the Neural Network will satisfy at best the physical constraint defined by the backwater model and the discrepancy with the data.

4.2 Build your PINN

Now that you know the expressions of the loss functions, you have to implement them in the code. They are to be implemented in the "Backwater_model.py" file. In a first time, you have to complete the J_{res} and the J_{BC} loss functions, respectively line 106 and 122. Then, once the PINN works for non-parametric inputs, you can complete the $J_{res\,parametric}$ loss function line 115 for the parametric inputs case !

4.3 Analyze your results

Now your PINN is working fine ! How sensitive is the approximation of the solution to the following hyper-parameters ?

Examples of hyperparameters :

- Number of colocation points N_{col} (can be modified at inputs generation),
- Number of hidden layers and number of neurons in each hidden layer (can be modified at model initialization),
- Choice of activation function (can be modified in the Class_PINN.py file, line 30),
- Training set size to testing set size ratio (can be modified at inputs initialization),
- Values of λ_{res} and λ_{BC} (can be modified in the Class_PINN.py file, line 130 for pre-training and line 149 for training),
- Number of iterations of the pre-training the training (can be modified at model training),
- Method for $\boldsymbol{\theta}_0$ at initialization (can be modified in the Class_PINN.py file, line 46-47, 53-54 and 59),

Choose some hyperparameters in the list (not all !) that you wish to deepen your understanding about and try to evaluate their influence on the results ! To properly measure the influence of one hyperparameter while keeping all the others constants, don't forget to use a seed when creating the model, the inputs or the observations ! This can be done in the main.py file, by adding seed = (an integer number).