Physics-informed neural networks for data-driven simulation: advantages, limitations, and opportunities

Félix Fernández de la Mata[a], Alfonso Gijón[a], Miguel Molina-Solana[a,b,*], Juan Gómez-Romero[a]

[a] *Dept. Computer Science and AI, Universidad de Granada, Spain*
[b] *Dept. Computing, Imperial College London, United Kingdom*

Abstract

The last decade has seen a rise in the number and variety of techniques available for data-driven simulation of physical phenomena. One of the most promising approaches is *Physics-Informed Neural Networks* (PINNs), which can combine both data, obtained from sensors or numerical solvers, and physics knowledge, expressed as partial differential equations. In this work, we investigated the suitability of PINNs to replace current available numerical methods for physics simulations. Although the PINN approach is general and independent of the complexity of the underlying physics equations, a selection of typical heat transfer and fluid dynamics problems was proposed and multiple PINNs were comprehensibly trained and tested to solve them. When PINNs were used as learned simulators, the outcome of our experiments was not entirely satisfactory as not enough accuracy was achieved even though optimal configurations and long training times were used. The main cause for this limitation was found to be the lack of adequate activation functions and specialized architectures, since they proved to have a notable impact on the final accuracy of each model. In turn, PINN architectures showed an accurate behavior when used for parameter inference of partial differential equations from data.

Keywords: Deep Learning, Physics-Informed Neural Networks, Learned Simulators, Data-driven Simulations

1. Introduction

A simulation of a physical system produces a set of quantitative predictions of physical magnitudes from a given model and a simulation technique. In mesh-based dynamical simulations, each prediction is tied to a specific time and position coordinates, while particle-based simulations also include a particle tag or index. Building a simulation to describe a physical system or process is often a complex process. The first step is usually understanding the physical mechanisms behind it and expressing them in a quantitative way to construct the model. This is done by both gathering data through experimentation and by finding the mathematical equations (typically partial differential equations, or PDEs for short) that better explain the behavior of the phenomenon under study. The most complex task starts at this point, as said equations need to be solved. Analytical solutions are unaffordable for most real-world problems, so one has to introduce simplifying approximations and/or employ numerical methods. Some of the most widely-used methods in this regard are FDM [1, 2] (Finite Difference Method), FEM [3] (Finite Element Method), and FVM [4] (Finite Volume Method).

With the surge of deep learning techniques [5], a collection of new methods for simulating physical systems has gained momentum. Some of them are GNNs [6] (Graph Neural Networks), fPINNs [7] (Fractional Physics-informed neural networks), and PINNs [8] (Physics-informed neural networks), the latter being

*Corresponding author
Email addresses: felixfdlm@correo.ugr.es (Félix Fernández de la Mata), alfonso.gijon@ugr.es (Alfonso Gijón), miguelmolina@ugr.es (Miguel Molina-Solana), jgomez@decsai.ugr.es (Juan Gómez-Romero)

the object of study of this work. Recent studies have exploited this family of methods as differentiable models to construct learned simulators in different domains [9, 10, 11, 12, 13, 14, 15, 16]. The utility of data-driven methods has been clearly proven when enough data is available and a complete mathematical description of the phenomenon is not feasible [17, 18, 19, 20, 21]. However, it remains to be determined if they can achieve the necessary accuracy and robustness to replace standard solvers and simulators when the equations governing a system are completely known. The present work aims to explore the utility of PINNs to substitute simulators as finite element methods and to estimate unknown physical parameters from PDEs.

Unlike standard numerical methods, PINNs lack a mathematical basis that guarantees their convergence to reliable solutions. Moreover, numerous hyperparameters must be set before the training phase, resulting in many possible configurations. There is not yet much knowledge about the performance of each hyperparameter and the best configurations for PINN training. Furthermore, the lack of specialized architectures to process sequential data in current implementations of PINNs suggests that their potential remains largely unexplored yet. To alleviate this lack of research, we experimented with different kinds of physical problems, such as heat propagation or fluid dynamics. Specifically, we have analyzed the fitness of several PINN architectures when applied to simple heat transfer problems with up to three spatial dimensions, by exploring an ample space of hyperparameters. Subsequently, we have shown the usefulness of a PINN as an inverse tool to infer a physical parameter from the equations governing the system and simulation data.

We organize the paper as follows: Section 2 briefly introduces the main ideas behind PINNs, explaining their applications and their unique training method. Section 3 describes the experimentation setup, presenting the modeling problems solved through the experimentation, the methods used to procure validation data, the metrics used for evaluation purposes, and the hyperparameter grid explored throughout the process. Section 4 analyses the results, focusing on which hyperparameters had the most significant impact in accuracy. Lastly, Section 5 concludes with a discussion on the current state of PINNs and the tasks ahead for their improvement.

2. Background

2.1. Concept

PINNs were first proposed as a method to work with physics-related data [8], and are useful for regression and analysis purposes. The tool we used in this work to build PINNs is SciANN [22]. Other options for simplified PINN building are NVIDIA Modulus (previously known as SimNet) [23], IRDLnet [24], DeepXDE [25] or the recent PDEBench [26].

One of the most promising features of PINNS is the ability to solve supervised learning tasks subjected to physical laws or other constraints expressed as partial differential equations [8]. To achieve a good description of a given phenomenon, it is essential to define an appropriate mathematical model and a set of equations capable of capturing the important elements of a phenomenon. Once the physical model is defined and written through a set of equations, the main difficulty in applying a PINN is to find a suitable architecture and train the neural network with the available data. This approach is quite general, which is a clear advantage over standard simulations, where one commonly needs a more profound understanding of the specific system to discern if a technique is suitable or not for the problem at hand.

In turn, standard numerical methods usually allow us to estimate the accuracy of the predictions [3, 27] or improve the convergence level if there are enough computational resources (increasing the mesh density and/or size, number of basis functions, simulation time, etc.). However, the process required for training PINNs (and every deep learning-based method) is generally slow [28] and the way to obtain more accurate results need to be clarified, especially when the amount of data is limited. Therefore, with current state-of-the-art PINNs, we generally sacrifice accuracy in favor of implementation simplicity.

Additionally, PINNs can serve to solve inverse problems involving nonlinear partial differential equations [8]. In fact, PINNs can be used for parameter estimation to discover the partial differential equations governing a phenomenon from the analysis of available data. Obviously, the goodness of the result will depend on the amount and quality of the data. An example of this kind of capacity can be seen in the SciANN showcase article [22], where the vorticity coefficient for a Navier-Stokes system is predicted accurately.

2.2. Exploitation of PDE-encoded information.

In order to model a physical problem through a PINN, we must first infer the PDE system representing it. Once we have this information, the first step is building a mesh representing the space to be simulated. In this case, the mesh is simply an array of points with shape (N_p, N_d) distributed as evenly as possible along the simulation space, where N_p is the number of points or data and N_d is the number of dimensions (including space and time).

The second step is building the neural network representing the problem (usually a Deep Neural Network or DNN). The inputs of the PINN will be the variables of the problem (space and time), keeping in mind the order they have in the mesh, and the outputs will be the function or functions which are the solution to the PDE system. Suppose any information about the properties of the solution functions is available, taking into account physical symmetries or conservation laws. In that case, we should also input those features into the neural network structure, as it will significantly improve the convergence rate. For instance, if it is known that the solution of a problem u has the shape $log(v)$ for some v, we should model the network to output a function v, and then add a lambda layer that performs the log operation over its output. This idea of imposing conditions on the modeling function can also be extended for the derivatives of the solution.

The last step is preparing the neural network to receive the PDE data. This rather complex step is more thoroughly explained in [8]. Next, we will just provide a simplified example with the heat equation.

Suppose a neural network representing a solution $u = N_u(W; x, t)$ is set up, where W denotes the network's weights, and (x, t) the predictors. We want this network to approximate the heat equation, $\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x_i^2}$. To do so, we need a term in the loss function that measures the deviation from the predictions along the mesh with regard to the equation. Luckily, since neural networks are differentiable by definition, we can define our loss function as

$$\mathcal{L}(W) = \sum_{i \in I} || \frac{\partial u}{\partial t}(x_i, t_i) - \frac{\partial^2 u}{\partial x_i^2}(x_i, t_i) || \ , \tag{1}$$

where $\{(x_i, t_i) | i \in I\}$ is the set of points that makes up the mesh. If we need to add more conditions, we can simply stack these losses by summation to obtain a loss that fits our requirements. In practice, it is much more convenient to do any operations such as differentiation, multiplication, or summation through lambda layers placed after the output, and use a classical loss function for the network, the result being the same. The SciANN library builds the lambda layers of the model automatically from the PDEs; the user only has to correctly write those equations and mind nothing about the underlying implementation to include the corresponding terms into the loss function.

Once the network is built, we can start the training process by simply fitting the neural networks to the points of the mesh. If there is data available from previous experiments, the loss function can be modified to accept them as well.

2.3. Parameter inference using PINNs

As we have seen, PINNs allow us to learn a function that is a solution to a system of known PDEs. It is also possible to do the inverse process and adjust (or, in some cases, learn) the equations describing a phenomenon using available data.

For instance, if we were reviewing heat-related data, we would work with the general form of the heat equation $\frac{\partial u}{\partial t} = \alpha \Delta u$, where α is a material-specific constant which explains the rate at which heat flows. We can consider α as a learnable parameter of our model, as well as the neural network weights and biases. As the predictions for u become more accurate during the training process, α would eventually converge to the physical constant we are trying to learn. Further details about this methodology can be found in [8, 19].

3. Methods

This section describes our experimentation setup. We first present the general design and then explain 1) the proposed problems, 2) how validation data was obtained, 3) the metrics used for evaluation purposes, and 4) how the hyperparameter search was performed.

As previously discussed, many hyperparameters must be defined when training a PINN model. Our goal was to measure which ones have a more significant influence on prediction accuracy to find the best configuration for both regression and parameter inference purposes. To do so, we defined four different physical problems as PDE systems; three of them were used for regression and one for parameter inference. Using FreeFem [29], a finite element method tool, we obtained validation data for each problem. Afterward, we coded an experimentation loop using MLflow [30] for each problem, which accepts a vector of hyperparameters, trains a PINN using them, evaluates the results against validation data, and logs them into an MLflow database. The problems were solved following increasing difficulty order, and after each one, the results were analyzed to discard the worst parameters and add new ones. For the last problem about parameter inference, only the best configuration of hyperparameters found in the previous experiments was used. The computational resources available for this work were limited because all the calculations were performed with a desktop computer with an Intel Core i5-4460 processor, 16 GB RAM memory and NVIDIA GTX 960 graphic card.

3.1. Experimental setup

Here we will briefly introduce the problems proposed and solved by the PINNs in the spatial and temporal domains. They are ordered with growing complexity, with the dimensions of the problem increasing gradually (this is due to the mesh size needed to simulate a system, which is highly dependent on its dimensions).

3.1.1. First Problem: Heat propagation on a 1D rod

The first problem consists of the simulation of heat along a 1D rod of length 2 during 10 time units[1]. As initial conditions (at $t = 0$), the temperature of the rod follows a linear relation with the position, ranging from zero temperature at the origin to 5 temperature units on its right border, located at $x = 2$. The rod is isolated at the origin (the temperature can evolve freely in that position) but is coupled to a heat bath of 5 temperature units at the $x = 2$ position, so that the temperature remains fixed along the experiment at that position. The equations defining the problem are the following:

$$\begin{cases} u_t - \Delta u = 0 & \text{On} \quad \Omega \times T & \text{(2a)} \\ \nabla u \cdot \nu = 0 & \text{On} \quad \{x = 0\} \times T & \text{(2b)} \\ u = 5 & \text{On} \quad \{x = 2\} \times T & \text{(2c)} \\ u = \dfrac{5}{2} x & \text{On} \quad \Omega \times \{t = 0\} & \text{(2d)} \end{cases}$$

Where Ω and T are the spatial and temporal domains, u represents the temperature and the product $\nabla u \cdot \nu$ accounts for the heat loss along the normal direction of a border. This is a classical PDE problem, and it can be solved by hand [31]. The function solution can be exactly written as

$$u(x,t) = \sum_{n=0}^{\infty} \left\{ \exp\left[-\left(\frac{2n+1}{4}\pi \right)^2 t \right] \frac{-5}{2\left(\frac{(2n+1)\pi}{4} \right)^2} \cos\left(\frac{2n+1}{4}\pi x \right) \right\} + 5 \ . \tag{3}$$

A full plot of the solution function is shown in Figure 1.

3.1.2. Second Problem: Heat source on a 2D plate

The second problem consists of the simulation of heat on a 2D square plate of side 4 during 10 time units. The plate is isolated on its left and right sides, and its upper and lower sides remain at temperature 5 the whole time. At time 5, a heat source with maximum temperature 10 gradually turns on on its center. The equations defining the problem are the following:

[1] Physical magnitudes are assumed to be dimensionless to avoid working with units.

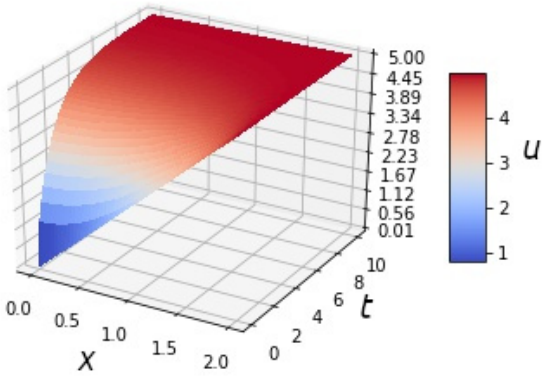

Figure 1: Analytical solution function of Equation 3, representing temperature u as a function of (x, t).

$$\begin{cases} u_t - \Delta u = f & \text{On} \quad \Omega \times T & \text{(4a)} \\ \nabla u \cdot \nu = 0 & \text{On} \quad \{x = -2\} \times T \cup \{x = 2\} \times T & \text{(4b)} \\ u = 5 & \text{On} \quad \{y = -2\} \times T \cup \{y = 2\} \times T & \text{(4c)} \\ u(x, y) = \dfrac{5}{2}|y| & \text{On} \quad \Omega \times \{t = 0\} & \text{(4d)} \end{cases}$$

Where f represents a heat source and is defined by

$$f(x, y, t) = \frac{10 \exp(-\frac{x^2+y^2}{2\sigma^2})}{\sqrt{2\pi\sigma^2}} \frac{1}{1 + \exp(-4(t-6))} \ . \tag{5}$$

This function is a Gaussian distribution located at the center of the plate with standard deviation $\sigma = 0.4$, which starts turning on at time 5, and at time 7 it is already fully activated. This situation is tested on purpose, as PINNs tend to force continuity into their predictions, and an abrupt activation would lead to worse results.

The evolution of the temperature of the problem is shown in Figure 2, at times 0, 5, and 9. In this case, the analytical solution becomes too complex, and we take a numerical solution obtained with FEM as the validation result.

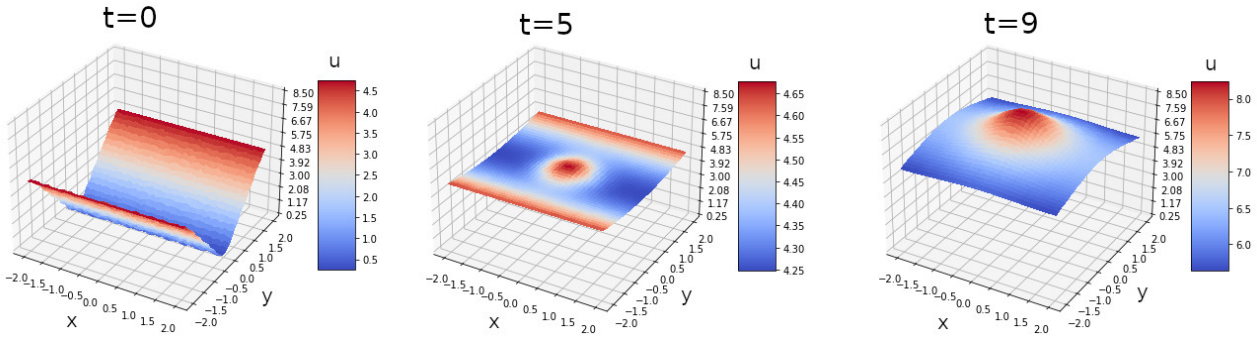

Figure 2: Numerical solution of Equation 4 obtained with FEM, representing the temperature on the vertical axis as a function of (x, y). Different panels refer to different times, in this case 0, 5 and 9 from left to right.

3.1.3. Third Problem: Heat well inside a 3D block

The third problem consists of the simulation of heat on a 3D cube of side 4 during 5 time units. As initial conditions (at $t = 0$), the whole cube has 30 temperature units. We will denote its top side as Γ_{top}, which will remain at temperature 30 for the whole experiment. We will also denote as Γ_{rest} the rest of the sides of the cube, which are isolated from the exterior.

$$\begin{cases} u_t - \Delta u = f & \text{On} \quad \Omega \times T & \text{(6a)} \\ \nabla u \cdot \nu = 0 & \text{Along} \quad \Gamma_{\text{rest}} \times T & \text{(6b)} \\ u = 30 & \text{Along} \quad \Gamma_{\text{top}} \times T & \text{(6c)} \\ u(x, y, z) = 30 & \text{On} \quad \Omega \times \{t = 0\} & \text{(6d)} \end{cases}$$

From the beginning of the experiment, a heat well described by

$$f(x, y, z) = -200 \exp(-\frac{x^2 + y^2 + z^2}{0.4^2}) \ , \tag{7}$$

reduces the temperature of the cube abruptly.

For better visualization of the heat well problem, Figure 3 shows the state of the problem at its beginning and its ending.

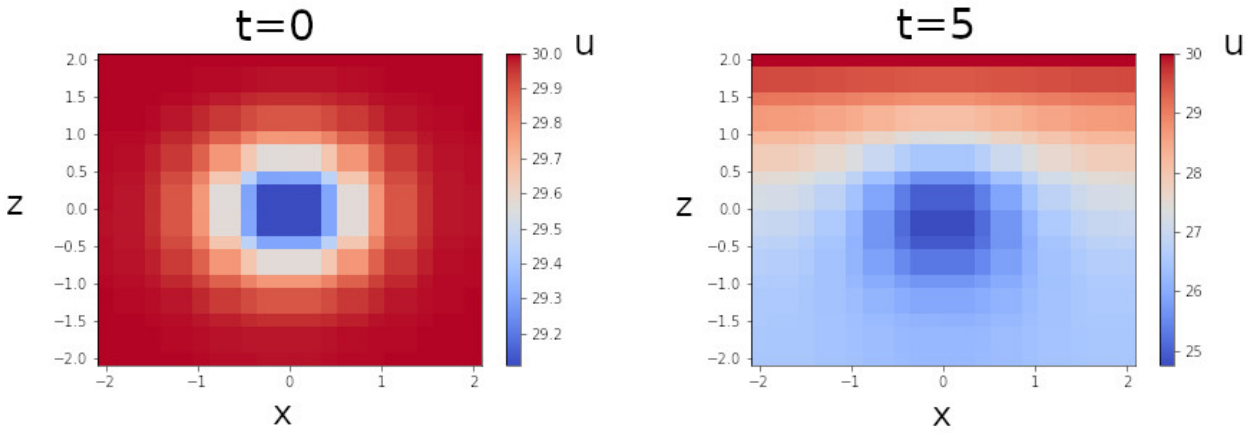

Figure 3: Representation of the temperature on slices of the cube along $y = 0$ at the beginning and ending of the problem. Numerical solution obtained with FEM.

3.1.4. Fourth Problem: Fluid Tunnel

This problem simulates the flow of a fluid through an irregular 2D tunnel during 30 time units. Our system consists of a fluid continuously entering a pipe from the left border and flowing towards the right border, see Figure 4. Initially, the fluid has a flat velocity profile, but the pipe becomes wider at a certain point, causing the appearance of a turbulent profile.

In this case, our goal is not to obtain a good prediction model for the problem but to attempt to predict the Reynolds number of the system. To do so, we use the best configuration found in previous problems and validation data obtained from a FEM calculation, along with the equations governing the problem, which also fed the PINN.

The code used to obtain the reference numerical FEM solution was taken from [29]. We introduced here a slight difference, the Reynolds number was decreased to 5×10^{-4} in order to increase the vorticity of the fluid, so to increase the complexity of the problem. The equations describing this problem are the following:

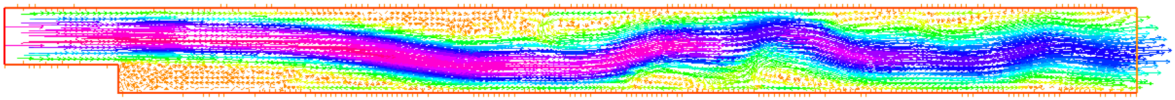

Figure 4: FEM prediction for Fluid Tunnel at $t = 30$.

$$
\begin{cases}
\partial_t u + u u_x + v u_y - u_{xx} - u_{yy} + p_x = 0 & \text{On} \quad \Omega \times T & \text{(8a)} \\
\partial_t v + u v_x + v v_y - v_{xx} - v_{yy} + p_y = 0 & \text{On} \quad \Omega \times T & \text{(8b)} \\
u_x + v_y = 0 & \text{On} \quad \Omega \times T & \text{(8c)} \\
u, v, p = 0 & \text{On } \Omega \times \{t = 0\} & \text{(8d)} \\
u = 4y(1-y), v = 0 & \text{On } \Gamma_1 \times T & \text{(8e)} \\
u\big|_{\Gamma_2} = u_{\Gamma_2}, v\big|_{\Gamma_2} = v_{\Gamma_2} & \text{On } \Gamma_2 \times T & \text{(8f)} \\
p = 0, v_{\Gamma_3} = 0 & \text{On } \Gamma_3 \times T & \text{(8g)}
\end{cases}
$$

The first three equations are the Navier-Stokes equations applied to a bi-dimensional problem, and the last four describe the initial state of the problem and where the borders, entrance, and output are located. Variables u and v represent the horizontal and vertical components of the velocity, p is the pressure, $\Gamma 1$ is the left border (input), $\Gamma 2$ accounts for the top and bottom borders of the pipe, and $\Gamma 3$ is the right border (output). For a better understanding of the setup, Figure 4 shows the state of the problem at time $t = 30$, after full flow across the tunnel is reached. The independent variables are (x, y, t) and the solution variables are (p, ψ), where velocities are defined as $u = \psi_y$ and $v = -\psi_x$, as discussed in [22].

3.2. Acquisition of validation data

Validation data for performance evaluation was obtained using the finite element analysis software FreeFem [29]. To do so, both the graph produced by FreeFem and the values predicted by it for each experiment were stored in different files for each experiment. Since the data produced by FreeFem does not necessarily match the mesh used on each experiment to train the PINN, we approximated the values on each mesh point using the interpolation utility from SciPy [32]. Spatial and temporal coordinates were treated equally to build the training mesh to solve the PDEs, so that an equal number of points per unit of distance and time were selected. The density of the mesh used to train the PINNS and other model details will be specified later in section 4.

3.3. Metrics used for performance measuring

In order to evaluate the performance of each hyperparameter configuration, validation predictions were compared to each PINN's predictions using various metrics. The norms L^1 and L^2 were chosen to measure the differences between the predictions and the validation dataset. These norms (and L^p norms in general) measure the distance between two functions over a given vector space, in this case the spatial-temporal $\Omega \times T$. Since analytical integration is not possible for the data we are working with, a numerical approximation was performed.

Considering that all training meshes consist of a set of points evenly distributed through the experimentation space, we can assume that a discretized approximation of the error is

$$
\int_\Omega |u_{\text{true}}(x) - u_{\text{pred}}(x)| dx \approx \sum_{i=1}^{N_p} |u_{\text{true},i} - u_{\text{pred},i}| \frac{\int_\Omega dx}{N_p} = \left[\frac{1}{N_p} \sum_{i=1}^{N_p} |u_{\text{true},i} - u_{\text{pred},i}| \right] \text{Vol}(\Omega) \ , \tag{9}
$$

where $\text{Vol}(\Omega)$ represents the volume of the region Ω. Extending this formula results in the following for L^1 and L^2:

$$L^1(f-g) = \int_{\Omega\times T} |f(x) - g(x)| dx \approx MAE(f,g) \int_{\Omega\times T} dx \tag{10}$$

$$L^2(f-g) = \int_{\Omega\times T} (f(x) - g(x))^2 dx \approx MSE(f,g) \int_{\Omega\times T} dx \tag{11}$$

Of course, as the training mesh becomes denser, this approximation becomes more accurate, which was taken into account to choose dense enough meshes for each problem within this work.

Another chosen metric is the maximum absolute error. Combined with the previous metrics, it is helpful to determine whether the error is evenly distributed or localized. Furthermore, training and testing times were also measured to compare the speed to achieve an accurate deep-learning approximation and a numerical one.

3.4. Hyperparameter search methodology

The hyperparameters we considered for the experimentation were: `mesh_density`, `activation_function`, `loss_function`, `optimizer`, `batch_size`, `epochs_of_training`, and `structure_of_the_neural_network`.

1. `mesh_density` determines the amount of points in the mesh per coordinate and volume unit (including time). For instance, for the 1D problem, independent variables are (x, t), so that the mesh is 2-dimensional. Therefore, a `mesh_density`=3 means each volume unit (area=distance×time in this case) has 3^2 points. This hyperparameter ranged between 5 and 50 in the experimentation, although its maximum was decreased as dimensionality increased.
2. `activation_function` defines the activation function each PINN uses for all of its layers. Possible values for this hyperparameter were `ReLU`, `Tanh`, `eLU`, `softplus`, `sigmoid`, `softmax`.
3. `loss_function` defines the function used to calculate the loss of each prediction. The only values used were `mae` and `mse`.
4. `optimizer` selects the optimizer used during training. Its main function is managing the weights and learning rate of the neural network during training in order to improve convergence rate. The used values were `Adam`, `Ftrl`, `Nadam`, `RMSprop` and `SGD`.
5. `batch_size` fixes the amount of samples given to the neural network each time. During the experimentation, `batch_size` ranged from 5000 to 25000.
6. `epochs_of_training` defines the amount of times the whole dataset fed the neural network. In the experimentation it ranged from 200 to 2000.
7. `structure_of_the_neural_network` defines the number of layers and amount of neurons per layer the neural network has. The multiple structures used in the experiments may be classified into three groups: few layers with many neurons, equal number of layers and neurons, and many layers with few neurons.

Since the hyperparameter grid of $\sim 10^4$ possible configurations is too large, experimentation on each problem was divided into two steps. Firstly, all possible combinations of mesh density, activation function, loss function, and optimizer were tested while fixing the rest of the hyperparameters. Secondly, batch size, epochs, and structure were searched while fixing the rest of the parameters on the optimal configuration. The proposed problems were addressed sequentially, following the order they were presented (1D, 2D, and 3D heat problems). At the beginning of each problem, the worst hyperparameters for the previous case were discarded to reduce the size of the hyperparameter grid to $\sim 3 \times 10^3$ and $\sim 2 \times 10^3$ possible combinations for the 2D and 3D systems, respectively. As commented previously, for the last problem (parameter inference), we used the insights from preceding experiments to select optimal hyperparameters, subject to our computational limitations.

4. Results and Discussion

In this section we analyze the outcome of the experimentation and comment on the behavior of the PINNs regarding the hyperparameters. As previously indicated, our first three problems (related to heat transfer) served to find the more suitable hyperparameters for the architecture of the PINNs; the last one (corresponding to fluid flow) provided the solution of a physical coefficient by using a predefined architecture and data from a FEM simulation.

Our study showed that some hyperparameters, such as `optimizer`, `loss_function`, and `batch_size`, did not have a significant effect on the outcome. Smaller values of `batch_size` had a slightly better accuracy result, and optimizers such as `Adam` and `SGD` gave better results, but their effect was essentially negligible. In contrast, the number of `epochs` and the training mesh density did have a strong expectable effect on the outcome, as they determine the amount of data that is passed to the network during the training process. The most interesting hyperparameters were `structure_of_the_network` and the `activation` function. The best network structures were the flat-shaped ones (a combination of many neurons per layer with a small number of layers), an expected result given that this kind of structure allows the backpropagation algorithm to achieve better results faster.

Regarding `activation` functions, we detected early in the first problem a significant gap in accuracy depending on the chosen activation function. In Figure 5, we plot L1 versus MaxError estimators for different model architectures and distinguishing among different activation functions represented with different colors. Different architectures and hyperparameters provide very different solutions, thus explaining the disparity of validation errors among different points. On the left panel (1D problem), we can see a wedge formed for some activation functions, such as `Softmax` and `Sigmoid`, in gray color. In fact, these activation functions are related to classification problems, and a deeper analysis showed that these models predict an almost constant temperature value for all the input data. That is, they apparently can follow the heat equation (a constant function is a solution of Equation 2a) but cannot learn from the data. For that reason, `Softmax` and `Sigmoid` functions were discarded in the first stage of the experimentation and they were not taken into account for the following problems in higher dimensions. For their part, activation functions related to regression problems, such as `ReLU` or `Tanh`, gave better results, with lower L1 and MaxError. In turn, hyperbolic tangent was the best among all the experiments, as can be observed in the center and right panels of Figure 5. `ReLU` produced worse results than `Tanh` or `eLU`, but trained much faster than the rest, achieving half the mean training time than `Tanh` for the 3D problem.

Despite these findings, no activation function gave reliable predictions for the plate or cube problems. They all resulted in L1, L2, and maximum absolute error values too large to be useful. Furthermore, the computational resources needed to achieve the best predictions were too large, requiring considerable training times and dense training meshes to achieve significant accuracy. Table 1 shows the optimal hyperparameters found for each problem, along with the employed mesh density, computational time, and metric details.

Contrary to the experiments devoted to solving a system of PDEs, the parameter inference experiment on a fluid tunnel yielded good results. Although the solution functions predicted by the PINN for u, v,

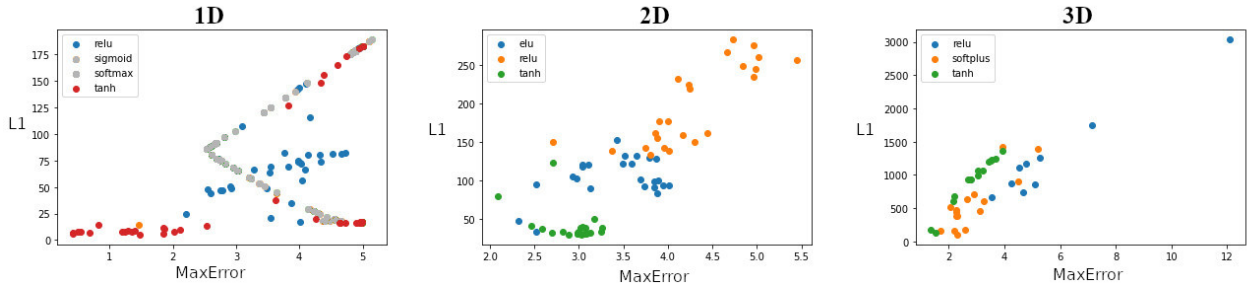

Figure 5: L1 vs MaxError for different activation functions on each one of the 3 first problems, going from 1D to 3D from left to right.

	1D rod	2D plate	3D cube	2D Fluid
`mess_density`	50	4	4	9
`activation`	`tanh`	`tanh`	`tanh`	`tanh`
`loss`	`mse`	`mae`	`mae`	`mae`
`optimizer`	`Adam`	`Adam`	`Adam`	`Adam`
`batch_size`	500	5000	5000	10000
`epochs`	2500	1000	1000	2500
`structure_NN`	3[100]	3[100]	3[200]	2 * 3[64]
L1	0.52	24.45	50.07	-
L2	0.78	38.62	78.49	-
MaxError	0.26	0.91	1.16	-
Time (min)	11.93	4.97	33.8	125

Table 1: Optimal hyperparameters and PINN architecture found for each problem.

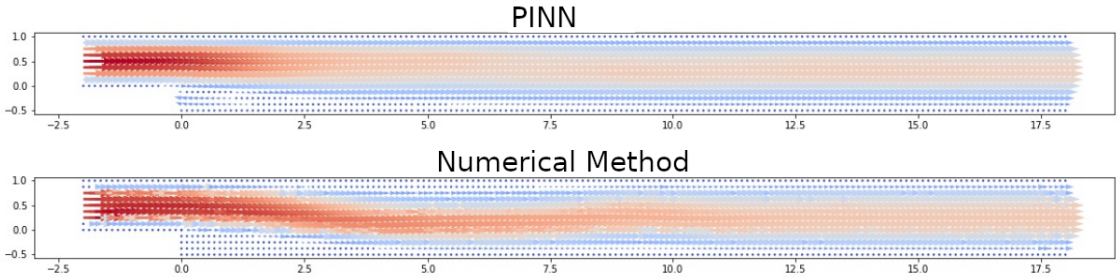

Figure 6: Comparison between the PINN prediction for the 2D fluid and the FEM numerical method at $t = 30$.

and p did not have great accuracy (see Figure 6), the Reynolds number predicted using the highest mess density and number of neurons was 5.14×10^{-4}, which is very close to its real value of 5×10^{-4}. As can be observed in Table 1, we used optimal hyperparameters for `activation`, `loss`, and `optimizer`, while the rest of parameters were selected to produce a neural network model as flexible as possible, always within the constraints of our computational resources. As mentioned before, two neural network solutions are necessary in this case, accounting for p and ψ, the velocities being derivatives of the latter one. Our tests with different hyperparameters and initializations of the neural network predicted approximately equal values for the inferred parameter (see Table 2), suggesting that this is a robust method.

5. Conclusions and Future Work

Regarding the capability of PINNs to replace current numerical methods for simulations, the problems proposed here were relatively simple for current available numerical solvers. As previously stated, standard numerical solvers are accurate and fast, although usually challenging to implement and adapt to a specific problem. Although all of the PINNs trained during this experimentation were quite inaccurate and required a lot of computational resources to provide good enough predictions, PINNs are simple to implement using a library such as SciANN [22], as only knowledge of the PDE system to be solved is needed to build one. Nonetheless, we did not find an adequate enough compromise between simplicity and accuracy, so we need to conclude that PINNs are not a good substitute for numerical solvers at this moment. The main obstacle to achieving more accurate predictions appears to be the lack of activation functions suitable for this kind of task, as they highly determine the accuracy of the predictions. Further experimentation expanding the set of activation functions tested should be performed in future works.

Our last point of interest regarding using PINNs as regressors is the need for specialized architectures for dealing with sequential data. The current structure of PINN training only allows for predictions to

	Case 1	Case 2	Case 3
mess_density	9	9	**8**
activation	tanh	tanh	tanh
loss	mae	mae	mae
optimizer	Adam	Adam	Adam
batch_size	10000	10000	10000
epochs	2500	2500	2500
structure_NN	2*3[64]	**2*3[60]**	2*3[64]
Reynolds number ($\times 10^4$)	5.14	4.7	5.6

Table 2: Comparison of the inferred Reynolds number for models with slightly different hyperparameters.

be constrained in space and time, that is, following the ideas of classical modeling. Applying the ideas behind PINNs to sequence-processing architectures —such as convolutional, recurrent, or attention-based networks— in order to avoid such constraints should also be an objective to be pursued in future research. Thus, the utility of a PINN as an accurate substitute for current simulators in the general case still requires investigation.

Concerning the parameter inference problem, it seems essential to remember that the data used to feed the PINN in our example would be difficult to measure from an experimental point of view, since data on such a dense mesh looks troublesome to gather. We observed good stability on the value of the inferred parameter, even though the accuracy of the solution functions for the PDEs provided by the PINN could have been more satisfactory. This result (on the appropriateness of PINNs when we know data but not the entire physics) is very promising, but further experimentation is required to check the method's robustness for different hyperparameters and net initializations, employing coarser and more realistic data.

Acknowledgments

This work was partially funded by FEDER/Junta de Andalucía (DEEPSIM project, A-TIC-244-UGR20), Spanish Ministry of Science (SINERGY project, PID2021-125537NA-I00) and the NextGenerationEU funds (IA4TES project, MIA.2021.M04.0008)

References

[1] J. L. Randall, Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems, Choice Reviews Online 45. doi:10.5860/choice.45-4438.

[2] J. C. Strikwerda, Finite Difference Schemes and Partial Differential Equations, Society for Industrial and Applied Mathematics, 2004. doi:10.1137/1.9780898717938.

[3] J. E. Akin, Finite element analysis with error estimators: An introduction to the FEM and adaptive error analysis for engineering students, Butterworth-Heinemann, 2005.

[4] R. Eymard, T. Gallouët, R. Herbin, Finite Volume Methods, in: J. Lions, P. Ciarlet (Eds.), Solution of Equation in Rn (Part 3), Techniques of Scientific Computing (Part 3), Vol. 7 of Handbook of Numerical Analysis, Elsevier, 2000, pp. 713–1018. doi:10.1016/S1570-8659(00)07005-8.

[5] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444. doi:10.1038/nature14539.

[6] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, P. W. Battaglia, Learning mesh-based simulation with graph networks (2021). arXiv:2010.03409, doi:10.48550/arXiv.2010.03409.

[7] G. Pang, L. Lu, G. E. Karniadakis, fPINNs: Fractional Physics-Informed Neural Networks, SIAM Journal on Scientific Computing 41 (4) (2019) A2603–A2626. doi:10.1137/18M1229845.

[8] M. Raissi, P. Perdikaris, G. 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 (2019) 686–707. doi:10.1016/j.jcp.2018.10.045.

[9] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, L. Yang, Physics-informed machine learning, Nature Reviews Physics 3 (6) (2021) 422–440. doi:0.1038/s42254-021-00314-5.

[10] H. Wessels, C. Weißenfels, P. Wriggers, The neural particle method – an updated lagrangian physics informed neural network for computational fluid dynamics, Computer Methods in Applied Mechanics and Engineering 368 (2020) 113127. doi:10.1016/j.cma.2020.113127.

[11] S. So, T. Badloe, J. Noh, J. Bravo-Abad, J. Rho, Deep learning enabled inverse design in nanophotonics, Nanophotonics 9 (5) (2020) 1041–1057. doi:10.1515/nanoph-2019-0474.
[12] R. Zhang, Y. Liu, H. Sun, Physics-informed multi-lstm networks for metamodeling of nonlinear structures, Computer Methods in Applied Mechanics and Engineering 369. doi:10.1016/j.cma.2020.113226.
[13] D. Lucor, A. Agrawal, A. Sergent, Simple computational strategies for more effective physics-informed neural networks modeling of turbulent natural convection, Journal of Computational Physics 456. doi:10.1016/j.jcp.2022.111022.
[14] P. Pilva, A. Zareei, Learning time-dependent PDE solver using message passing graph neural networks (2022).
URL https://openreview.net/forum?id=oaKw-GmBZZ
[15] J. Klicpera, F. Becker, S. Günnemann, Gemnet: Universal directional graph neural networks for molecules, in: A. Beygelzimer, Y. Dauphin, P. Liang, J. W. Vaughan (Eds.), Advances in Neural Information Processing Systems, 2021.
URL https://openreview.net/forum?id=HS_sOaxS9K-
[16] C. W. Park, M. Kornbluth, J. Vandermause, C. Wolverton, B. Kozinsky, J. P. Mailoa, Accurate and scalable graph neural network force field and molecular dynamics with direct force architecture, npj Computational Materials 7 (1). doi:10.1038/s41524-021-00543-3.
[17] B. C. Daniels, I. Nemenman, Automated adaptive inference of phenomenological dynamical models, Nature Communications 6 (1). doi:10.1038/ncomms9133.
[18] S. L. Brunton, J. L. Proctor, J. N. Kutz, W. Bialek, Discovering governing equations from data by sparse identification of nonlinear dynamical systems, Proceedings of the National Academy of Sciences of the United States of America 113. doi:10.1073/pnas.1517384113.
[19] S. H. Rudy, S. L. Brunton, J. L. Proctor, J. N. Kutz, Data-driven discovery of partial differential equations, Science Advances 3 (4). doi:10.1126/sciadv.1602614.
[20] K. Champion, B. Lusch, J. N. Kutz, S. L. Brunton, Data-driven discovery of coordinates and governing equations, Proceedings of the National Academy of Sciences of the United States of America 116. doi:10.1073/pnas.1906995116.
[21] Z. Chen, Y. Liu, H. Sun, Physics-informed learning of governing equations from scarce data, Nature Communications 12 (1). doi:10.1038/s41467-021-26434-1.
[22] E. Haghighat, R. Juanes, SciANN: A Keras/TensorFlow wrapper for scientific computations and physics-informed deep learning using artificial neural networks, Computer Methods in Applied Mechanics and Engineering 373. doi:10.1016/j.cma.2020.113552.
[23] O. Hennigh, S. Narasimhan, M. A. Nabian, A. Subramaniam, K. Tangsali, M. Rietmann, J. d. A. Ferrandis, W. Byeon, Z. Fang, S. Choudhry, NVIDIA SimNet: an AI-accelerated multi-physics simulation framework (2020). doi:10.48550/ARXIV.2012.07938.
URL https://arxiv.org/abs/2012.07938
[24] W. Peng, J. Zhang, W. Zhou, X. Zhao, W. Yao, X. Chen, Idrlnet: A physics-informed neural network library (2021). arXiv:2107.04320.
[25] L. Lu, X. Meng, Z. Mao, G. E. Karniadakis, Deepxde: A deep learning library for solving differential equations, SIAM Review 63 (1) (2021) 208–228. doi:10.1137/19M1274067.
[26] M. Takamoto, T. Praditia, R. Leiteritz, D. MacKinlay, F. Alesiani, D. Pflüger, M. Niepert, PDEBench: An extensive benchmark for scientific machine learning, in: Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track, 2022.
URL https://openreview.net/forum?id=dh_MkX0QfrK
[27] W. L. Oberkampf, S. M. DeLand, B. M. Rutherford, K. V. Diegert, K. F. Alvin, Error and uncertainty in modeling and simulation, Reliability Engineering and System Safety 75. doi:10.1016/S0951-8320(01)00120-X.
[28] Y. Shin, J. Darbon, G. E. Karniadakis, On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type pdes, Communications in Computational Physics 28 (5) (2020) 2042–2074. doi:10.4208/cicp.OA-2020-0193.
[29] F. Hecht, New development in FreeFem++, Journal of Numerical Mathematics 20 (3-4) (2012) 251–265. doi:10.1515/jnum-2012-0013.
URL https://freefem.org/
[30] M. A. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, F. Xie, C. Zumar, Accelerating the Machine Learning Lifecycle with MLflow, IEEE Data Eng. Bull. 41 (2018) 39–45.
[31] L. C. Evans, Partial Differential Equations, 2nd Edition, Vol. 19 of Graduate Studies in Mathematics, American Mathematical Society, 2010.
[32] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, SciPy 1.0 Contributors, SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python, Nature Methods 17 (2020) 261–272. doi:10.1038/s41592-019-0686-2.