Data-Driven Parameter Discovery of a One-Dimensional Burgers' Equation Using a Physics-Informed Neural Network

1st Eduardo F. Miranda
Applied Computing (CAP)
National Institute for Space Research
São José dos Campos, Brazil
efurlanm@gmail.com

2nd Leonardo B. L. Santos
Applied Computing (CAP)
National Institute for Space Research
São José dos Campos, Brazil
santoslbl@gmail.com

3rd Stephan Stephany
Applied Computing (CAP)
National Institute for Space Research
São José dos Campos, Brazil
stephan.stephany@inpe.br

***Abstract*—This work demonstrates the use of a Physics-Informed Neural Network (PINN) trained to solve supervised learning tasks respecting the law of physics described by the one-dimensional Burgers partial differential equation (PDE), and focuses on the problem of data-driven PDE parameter discovery. The Burgers' equation is one PDE with derivatives in space and time that is commonly solved by a numerical method. However, recent work proposes the use of PINN to solve, as a new class of data-efficient universal function approximators, which naturally encode any underlying physical laws as prior information. As the number of sample points required for efficient Deep Natural Network (DNN) training would be very high, PINN was proposed, allowing the use of a smaller number of sample points, and incorporating the related physical equation in the simulation. This work evaluates the discovery of parameters of the Burgers' equation through the use of PINN, for different hyperparameters and dataset sizes, seeking the best adjustment. The relative errors and processing times obtained are presented, running on the LNCC's Santos Dumont supercomputer.**

I. Introduction

Many simulations are mathematically modeled by Partial Differential Equations (PDEs), which have derivatives in space and time. However, the coefficients of these derivatives are unknowns, and the PDEs are usually solved by a numerical method, like the finite difference method. Recent works proposed to solve PDEs using Deep Neural Networks (DNN), which are machine learning algorithms. The universal approximation theorem states that a neural network can approximate any continuous function, as long as the network has a sufficient number of hidden layers and employs nonlinear activation functions. This approach requires knowledge of a large set of sample points in space and time domain (called Collocation Points - CPs) to train the DNN, which can be obtained either by observation, or if the model is known, it can be generated by numerical methods. As the required number of CPs would be very high, Physics-Informed Neural Networks (PINNs) were proposed and allow the use of a smaller number of CPs as they include the underlying physical laws related to the simulation in the DNN.

PINNs can be used in direct problems (inference or solution), where the PDE and parameters are known and we want to obtain the simulation result, and in inverse problems (identification or discovery) where we have the dataset and want to obtain the PDE parameters. A work by Chevallier et al. [1] describes a speedup of 7 using DNN to obtain parameters in the Longwave Radiative Transfer model from ECMWF (European Center for Medium-Range Weather Forecasts), showing the importance of using DNN to obtain parametric representation in numerical modeling of various atmospheric processes. Krasnopolsky et al. [2] also cites speedups between 10 to 10^5 using DNN in the parametrization of physical models in oceanic and atmospheric numerical models. Furthermore, there is also the possibility of using PINN in cases where the model (or the PDE that describes it) is known, to reduce the size of the dataset necessary to train the DNN, thus increasing efficiency, or in cases where there is noise in the sample and we want the underlying physical law to help deal with it.

This work evaluates data-driven parameter discovery of the one-dimensional Burgers' equation, a PDE with derivatives in space and time, obtained through PINN, for different hyperparameters and dataset sizes. The work seeks to answer the question "what is the ideal combination of hyperparameters and dataset size, for this specific problem?", in order to seek the best model for the expected result.

The PINN discovery is evaluated in terms of accuracy and DNN training processing time, executed on the Santos Dumont supercomputer (SDumont) at the National Scientific Computing Laboratory (LNCC). The tests were carried out on a Bull Sequana X1120 processing node with two 2.1 GHz 24-core Intel Xeon Gold 6252 Skylake processors (totaling 48 cores), 384 GB of main RAM, and four Nvidia Volta V100 GPUs. Only one GPU is used in this work. All data and codes used in this manuscript are publicly available on GitHub at https://github.com/efurlanm/425/.

The discovery of EDPs by PINNs is relatively recent and the acquisition of knowledge in this approach can be useful for application in some specific modules of numerical models used at CPTEC/INPE for weather and climate prediction.

II. Material and methods

Raissi et al. [3] published an article about PINNs, which has 7217 citations (December 2023). That work defines PINNs as DNNs trained to solve supervised learning tasks, but complying to physical laws, usually described by nonlinear PDEs. It also describes the use of DNNs to solve PDEs and obtain physics-informed surrogates of the physical model that are fully differentiable in all coordinates and free parameters. PINNs form a new class of data-efficient universal function approximators, which can be effectively trained using small datasets, and which may encode any underlying physical law.

DNN training data can be randomly sampled from observational data, or through simulations using synthetic data from a numerical model. Except for synthetically generated data, as long as a sufficient number of CPs are available, a standard DNN can solve the PDE, otherwise a PINN would be required. A PINN uses a specific loss function incorporating PDE and parameters, in such a way that during the training phase using the set of CPs, the applicable physical law is incorporated [4].

PINNs can be considered neural networks for supervised learning problems, as proposed here. However, PINNs can also be used as agents for Reinforcement Learning (RL) [4]. The most common PINN architectures are Multi-layer Perceptrons (MLPs), Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). Newer architectures are Auto-Encoder (AE), Deep Belief Network (DBN), Generative Adversarial Network (GAN) and Bayesian Deep Learning (BDL) [4]. This work uses the MLP architecture.

The proposed test problem requires the parameters discovery of a particular one-dimensional Burgers' equation, which estimates the speed field u along time (Equation 1). Training data for the PINN is given by a set of CPs corresponding to the position field in different times are randomly generated within the considered domain.

In the train phase, the neural network then estimates a solution $u(t,x)$. The function employed by the PINN, $f(t,x)$ (Equation 2), is derived from the known Burgers' equation, and allows to calculate the loss function. The parameters of the differential operator that we want to obtain are transformed into PINN parameters. In the following equations, the differential operator parameter λ_1 (or u) is the speed of fluid at the indicated spatial and temporal coordinates, the differential operator parameter λ_2 (or ν) is the kinematic viscosity of fluid, and the subscripts denote partial differentiation in time and space, respectively, as u_t (which denotes $\frac{du}{dt}$), u_x (which denotes $\frac{du}{dx}$), and u_{xx} (which denotes $\frac{d^2u}{dx^2}$).

$$u_t + \lambda_1 u_x - \lambda_2 u_{xx} = 0, \quad x \in [-1,1],\ t \in [0,1] \tag{1}$$

The Burgers' equation is employed to evaluate the error f of the solution $u(t,x)$ estimated by the PINN, as shown in Equation 2.

$$f := u_t + \lambda_1 u_x - \lambda_2 u_{xx} \tag{2}$$

In this work, the PINN loss function to be minimized is given by the mean squared error (Equation 3) of two components, MSE_u, which embeds the training data on $u(t,x)$, and MSE_f, which embeds the structure imposed by Equation 1, where t is the time step, and x is the one-dimension coordinate. The neural network parameters, along with the differential operator parameters λ_1 and λ_2, can be learned by minimizing the MSE.

$$MSE = MSE_u + MSE_f \tag{3}$$

where

$$MSE_u = \frac{1}{N}\sum_{i=1}^{N} |u(t_u^i, x_u^i) - u^i|^2$$

and

$$MSE_f = \frac{1}{N}\sum_{i=1}^{N} |f(t_u^i, x_u^i)|^2$$

The $\{t_u^i, x_u^i, u^i\}_{i=1}^N$ denotes the training data on $u(t,x)$, the MSE_u loss corresponds to the training data in $u(t,x)$, and the MSE_f loss imposes the structure of Equation 1 on a finite set of CPs. The number and location of CPs are the same as the training data.

In this work a dataset of 2,000 points generated by the numerical Gaussian Quadrature Method (GQM), using $\lambda_1 = 1$ and $\lambda_2 = 0.01/\pi$, was used to obtain the CPs, that are also used to compare the result obtained through PINN. The GQM method is an iterative numerical algorithm that approximates the definite integral of a function as a weighted sum of the function values at specified points within the domain of integration [5].

When training a PINN, some important adjustable hyper-parameters are the number of hidden layers $N_l (l = 1, 2, ...)$, and the number of neurons in each layer $N_{le} (e = 1, 2, ...)$. A general understanding about N_l and N_{le} is that efficient adjustment is still an unsolved problem and the determination is made empirically [7].

The results obtained in this work using DNN are subject to the problem of overfitting and underfitting. Overfitting means that the DNN performs very well when using training data, but fails as soon as it needs to deal with new data in the problem domain, that is, it does not generalize. Underfitting, on the other hand, means that the model performs poorly on both datasets, i.e., it does not fill the model. Both issues can also negatively affect performance [6].

The Relative L2 Error used in this work is introduced here as defined in Equation 4 where $\|\widehat{U} - U\|$ is the L2 norm of the prediction deviation at certain time, and $\|U\|$ denotes the L2 norm of the synthetic data at that time. R_{L2} gives good quantification of the prediction accuracy at a certain time [7].

$$R_{L2} = \frac{\|\widehat{U} - U\|}{\|U\|} \tag{4}$$

A. PINN Implementation

The specific PINN architecture implemented in this work is an MLP network with an input layer of 2 neurons, a number of hidden layers ranging from 1 to 8, with each hidden layer having a number of neurons ranging from 10 to 30, and a output layer with one neuron. The loss function is the mean squared error (MSE). Minimization of the loss function is performed by an optimization method, the generalized limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) algorithm, a quasi-Newton method. All hidden layers employ the hyperbolic tangent as the activation function. The implementation has been configured to stop training when it reaches 50,000 iterations or when the hardware's floating point precision is interfering with the calculated error.

The PINN implementation is based on the work of Raissi et al. (2019) [3] and uses the TensorFlow[1] 1.15 library and the Python 3.7 interpreter. Code snippets of the TensorFlow library are shown in Listing 1 and Listing 2. The code was run on SDumont and uses a V100 GPU.

Listing 1. Code snippet that implements $u(t,x)$

```
def neural_net(self, X, weights, biases):
  num_layers = len(weights) + 1
  H = 2.0 * (X - self.lb) / (self.ub - self.lb) - 1.0
  for l in range(0, num_layers - 2):
    W = weights[l]
    b = biases[l]
    H = tf.tanh(tf.add(tf.matmul(H, W), b))
  W = weights[-1]
  b = biases[-1]
  Y = tf.add(tf.matmul(H, W), b)
  return Y

def net_u(self, x, t):
  u = self.neural_net(tf.concat([x, t], 1), self.weights,
     self.biases)
  return u
```

Listing 2. Code snippet that implements $f(t,x)$

```
def net_f(self, x, t):
  lambda_1 = self.lambda_1
  lambda_2 = tf.exp(self.lambda_2)
  u = self.net_u(x, t)
  u_t = tf.gradients(u, t)[0]
  u_x = tf.gradients(u, x)[0]
  u_xx = tf.gradients(u_x, x)[0]
  f = u_t + lambda_1 * u * u_x - lambda_2 * u_xx
  return f
```

To obtain the results, first the network is trained until the parameters are obtained, then the prediction is made and compared with the values of the training dataset, which is used both to train the network and compare the results. The implementation does not clearly divide the dataset into training, validation, and testing, however it would be an improvement to be investigated in future work.

III. Results

The following results are divided into 4 parts, in the last 3 the number of neurons per layer is fixed on the horizontal axis, and the vertical axis varies according to the number of layers and size of the dataset. In the first part the result is shown in the form of a graph rendered as a pseudo-colored image showing t, x and $u(t,x)$ (visual evaluation of the PINN's predictive accuracy), and also a slice at time $t = 0.5$. In the second part, the vertical axis is the number of layers ("Neurons x Layers" for simplicity), and in the third part it is the size of the dataset ("Neurons x Dataset" for simplicity). The fourth and final part shows the prediction time for some number of neurons per hidden layer and number of layers. The size of the dataset in some cases is called CP, meaning the same thing.

[1] http://www.tensorflow.org

A. Visual assessment

A visual assessment of PINN's predictive accuracy is shown in Figure 1, with time t on the horizontal axis and spatial coordinate x on the vertical axis. The color scale refers to the speed $u(x,t)$. The black marks on the graph represent 2,000 CPs randomly generated and used for training, and to obtain them $\lambda_1 = 1$ and $\lambda_2 = 0.01/\pi$ were used. The network architecture used is composed of 4 hidden layers with 20 neurons each. The white solid vertical line at $t = 0.5$ represents a specific snapshot shown in Figure 2, which shows the overlapping solutions for PINN and GQM. For this specific result, the equation obtained by PINN is $u_t + 0.99958u_x - 0.0032199u_{xx} = 0$ whereas the equation used to obtain the training dataset is $u_t + u_x - 0.0031831u_{xx} = 0$. The network is able to identify the underlying partial differential equation with remarkable accuracy.

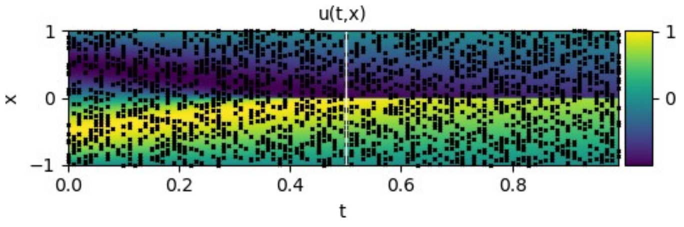

Fig. 1. Predicted solution u(t, x) along with the training data. The horizontal axis denotes time t, and the vertical axis, the coordinate x. The marks in the graph represent the randomly assigned CPs used for training. The color scale refers to the speed u(t, x). The solid white vertical line refers to the snapshot $t = 0.5$ shown in Figure 2.

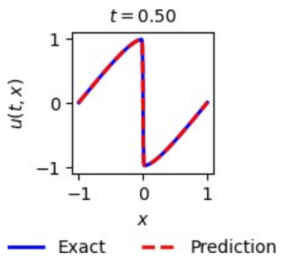

Fig. 2. Superimposed solutions for PINN (in red) and numerical solution (in blue) for the $t = 0.5$ snapshot.

B. Neurons x Layers

For the results presented below, the hyperparameters N_l (number of hidden layers) and N_{le} (number of neurons per hidden layer) were varied, as well as the number of CPs for training. The Table I shows the relative L2 errors and training times of the neural network, for different hyperparameters used:

10, 15, 20, 25, and 30 neurons per hidden layer, and 1, 2, 4, 6, and 8 hidden layers. The number of CPs was set at 2,000. All values shown here are the average of 3 runs. In this table it is possible to observe that there is a tendency for the best values to be concentrated in the center, probably because there is a problem of underfitting or overfitting in the values at the edges of the table. One of the highlights is that the smallest error is obtained with 6 hidden layers, not 8. In this specific case, increasing the number of layers not only does not increase precision, but also worsens performance.

Hidden layers	Number of neurons per hidden layer				
	10	15	20	25	30
	Relative L2 Error (%)				
1	18.54	17.77	17.80	17.53	17.47
2	3.70	2.52	1.52	1.77	1.59
4	0.30	0.41	0.44	0.39	0.16
6	0.22	0.19	0.10	0.18	0.17
8	0.26	0.13	0.19	0.16	0.23
	Training - processing time (seconds)				
1	4.2	5.4	5.0	21.7	9.4
2	35.9	51.8	39.3	55.5	70.7
4	51.2	43.1	33.4	40.9	47.4
6	59.7	40.3	42.5	35.2	38.6
8	58.7	60.0	58.6	54.4	84.5

TABLE I
RELATIVE L2 ERRORS AND DNN TRAINING TIMES FOR DIFFERENT NUMBER OF NEURONS AND HIDDEN LAYERS. ON THE COLOR SCALE, THE BEST VALUES ARE HIGHLIGHTED IN RED.

The Figure 3 shows that the error for 1 hidden layer is high compared to the other number of layers. For 2 layers there is a significant improvement in accuracy. For 4, 6, and 8 the gain in precision is not that great, but the curves are similar and are in the region of greater precision, showing that they would be the best choices.

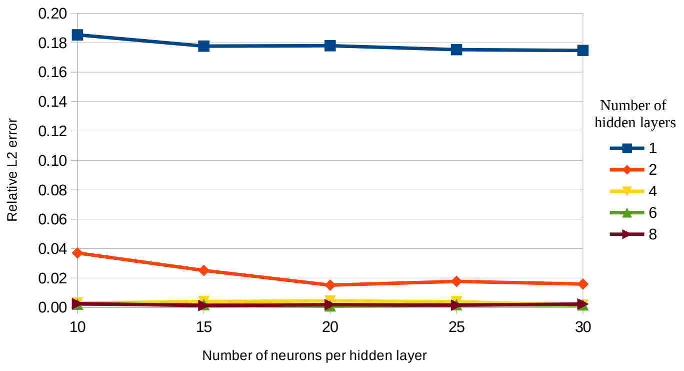

Fig. 3. Relative L2 error (%) in function of number of neurons and hidden layers.

The Figure 4 shows for 4 hidden layers, a tendency to describe a curve that resembles a parabolic, with a minimum processing time of 20 neurons per hidden layer. This is probably due to the problem of underfitting and overfitting occurring at the beginning and at the end of the curve.

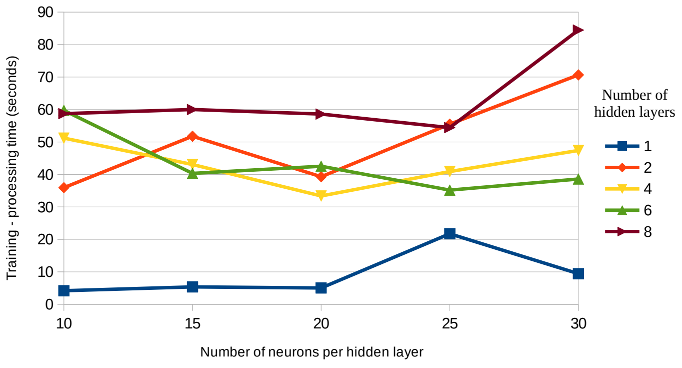

Fig. 4. Processing times (seconds) in function of number of neurons and hidden layers.

C. Neurons x Dataset

The table Table II shows the relative L2 errors and training times of the neural network, for different hyperparameters and number of CPs used: 10, 15, 20, 25, and 30 neurons per hidden layer, and 400, 800, 1200, 1600, and 2000 CPs. The number of layers was set at 8. All values shown here are the average of 3 runs. In this table, as in the previous one, it is possible to observe that there is a tendency for the best values to be concentrated in the center, probably because the problem of underfitting or overfitting is occurring in the values at the edges of the table. One of the highlights is that considering the smallest error and the shortest processing time, the best dataset size is 1600, and the best number of neurons per hidden layer is 20.

Dataset size	Number of neurons per hidden layer				
	10	15	20	25	30
	Relative L2 Error (%)				
400	3.12	3.30	2.83	1.84	6.36
800	1.79	0.83	0.59	0.52	0.34
1200	0.41	0.50	0.46	0.35	0.61
1600	0.90	0.51	0.19	0.46	0.13
2000	0.26	0.13	0.19	0.16	0.23
	Training - processing time (seconds)				
400	57.9	82.3	83.3	59.8	58.6
800	79.7	53.5	63.2	45.0	63.0
1200	63.7	52.2	43.8	42.1	56.8
1600	59.9	27.5	45.3	46.5	56.4
2000	58.7	60.0	58.6	54.4	84.5

TABLE II
RELATIVE L2 ERRORS AND DNN TRAINING TIMES FOR DIFFERENT NUMBER OF NEURONS AND DATASET SIZE. THE NUMBER OF HIDDEN LAYERS IS SET TO 8. ON THE COLOR SCALE, THE BEST VALUES ARE HIGHLIGHTED IN RED.

The Figure 5 shows that the error for 400 CPs is high compared to the others, 800 CPs presents a significant improvement in precision, and the other curves are relatively close, not presenting such a large accuracy gain.

The Figure 6 shows for most curves a tendency to describe a curve that resembles a parabolic, probably due to the problem

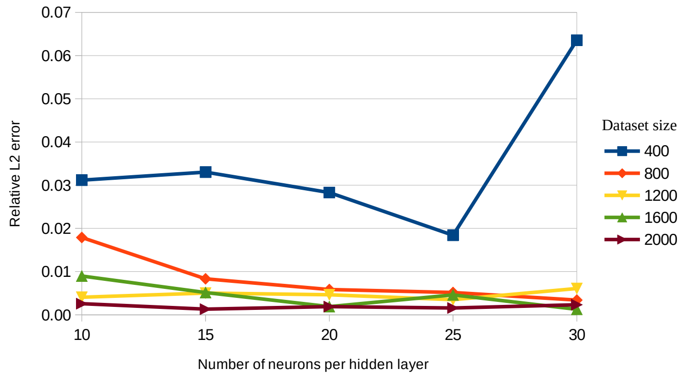

Fig. 5. Relative L2 error (%) in function of number of neurons and dataset size. The number of hidden layers is set to 8.

of underfitting and overfitting occurring at the beginning and end of the curve. The shortest processing time occurs for 15 neurons per hidden layer, and 1600 CPs.

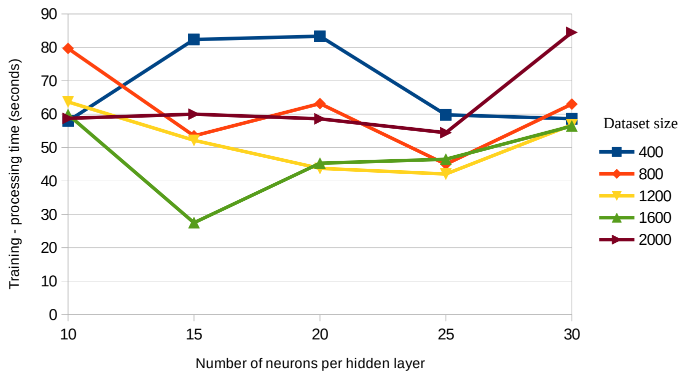

Fig. 6. Processing times (seconds) in function of number of neurons and dataset size. The number of hidden layers is set to 8.

D. *Prediction time*

The Table III shows the neural network's prediction times, once training is complete. Times are for 10, 20, and 30 neurons per hidden layer, and 1, 4, and 8 layers. Most times are relatively close, around 0.7 seconds. Compared to the training time of about 43 seconds in the best cases, the time to predict the final result represents about 1.5% of the training time, relatively. The difference is very large, and shows that we should look for algorithms or solutions where the number of trainings is smaller than the number of predictions, when applicable.

Number of hidden layers	Number of neurons per hidden layer		
	10	20	30
1	0.647	0.636	0.675
4	0.704	0.724	0.705
8	1.092	0.867	0.789

TABLE III
PREDICTION TIMES FOR DIFFERENT NUMBER OF NEURONS AND HIDDEN LAYERS. ON THE COLOR SCALE, THE BEST VALUES ARE HIGHLIGHTED IN RED.

IV. CONCLUSIONS

This work evaluates data-driven parameter discovery for a one-dimensional Burgers' equation using a Physics-Informed Neural Network (PINN). The Burgers' equation is a fundamental partial differential equation (PDE) with derivatives in space and time, which is commonly solved by a numerical method. An evaluation of the relative error and required training time, performed in SDumont, is also presented for different hyperparameters and dataset sizes. It was possible to observe that adjusting the hyperparameters and the size of the dataset is important for obtaining performance when using PINN. The implementation also proved to be relatively simple, and the results easy to obtain. As deep learning technology continues to grow rapidly, both in terms of methodological and algorithmic developments, this could be a timely contribution that can benefit a wide range of scientific domains. As future work, it would be interesting to explore other PINN architectures, as well as taking advantage of the parallel use of GPU.

ACKNOWLEDGMENT

Authors thank LNCC (National Laboratory for Scientific Computing) for grant 205341 AMPEMI (call 2020-I), which allows access to the Santos Dumont supercomputer (node of the SINAPAD, the Brazilian HPC system).

REFERENCES

[1] F. Chevallier, J. Morcrette, F. Chéruy, and N. A. Scott, "Use of a neural-network-based long-wave radiative-transfer scheme in the ECMWF atmospheric model," *Quart J Royal Meteoro Soc*, vol. 126, no. 563, pp. 761–776, Jan. 2000, {06}. [Online]. Available: https://rmets.onlinelibrary.wiley.com/doi/10.1002/qj.49712656318

[2] V. M. Krasnopolsky and M. S. Fox-Rabinovitz, "A new synergetic paradigm in environmental numerical modeling: Hybrid models combining deterministic and machine learning components," *Ecological Modelling*, vol. 191, no. 1, pp. 5–18, Jan. 2006, {04}. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304380005003455

[3] M. Raissi, P. Perdikaris, and 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*, vol. 378, pp. 686–707, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021999118307125

[4] S. Cuomo, V. S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli, "Scientific Machine Learning Through Physics-Informed Neural Networks: Where we are and What's Next," *arXiv preprint arXiv:2201.05624*, 2022. [Online]. Available: https://arxiv.org/abs/2201.05624

[5] J. Burkardt, "Investigating Uncertain Parameters in the Burgers Equation," Mathematics Department, Ajou University, Suwon, Korea, 2013. [Online]. Available: https://people.sc.fsu.edu/~jburkardt/presentations/burgers_2013_ajou.pdf

[6] W. Koehrsen, "Overfitting vs. underfitting: A complete example," *Towards Data Science*, vol. 405, 2018. [Online]. Available: http://www.pstu.ac.bd/files/materials/1566949131.pdf

[7] S. Xu, Z. Sun, R. Huang, G. Dilong, G. Yang, and S. Ju, "A practical approach to flow field reconstruction with sparse or incomplete data through physics informed neural network," *Acta Mechanica Sinica*, vol. 39, Nov. 2022.