Comparison of Physics-Informed and Classical Neural Networks in Approximating Nonlinear Pendulum Dynamics

Predrag Nikolić, Pavle Stepanić
Computer-Controlled Systems
Lola Institute Ltd
Belgrade, Serbia
predrag.nikolic@li.rs, pavle.stepanic@li.rs

Abstract—In this work, we will analyze and compare the approximation performance of physics-informed neural networks (PINNs) with that of ordinary neural networks (NNs) using numerically generated solutions of simple damped nonlinear pendulum dynamics. To make a fair comparison between the two network types, we will use linear and random uniformly distributed data points to test and analyze the results. This work emphasizes the key differences between traditional NNs and PINNs, analyzes their performance, and draws relevant conclusions.

Keywords— Machine learning, NNs, PINNs, Nonlinear dynamics, Predictive modeling

I. Introduction

The introduction of Physics-Informed Neural Networks (PINNs) marks a significant shift in the application of machine learning techniques to scientific computing and the newly established domain of scientific machine learning (SciML), particularly for solving complex partial differential equation (PDE) - driven problems. PINNs fundamentally differ from conventional neural networks, which are highly dependent on large datasets. Instead of looking for a lot of data to learn the correct patterns, PINNs recover the missing information from physical laws. The math behind these laws is integrated into the network's structure and becomes its main source. This extension enables PINNs to approximate physical phenomena with significantly greater accuracy, even with scarce, absent, or low quality data [1].

Recent studies have demonstrated the strength of PINNs across various applications, notably in modeling wave physics [2], fluid dynamics [3], and cardiac activation mapping [4]. For example, the work of Sa, J. et al. [5] highlighted how PINNs can accurately predict velocity fields and shear-stress responses across different fluid systems. In addition, they showed essential fluid features even when the data is limited or contaminated with noise. Furthermore, researchers have proposed adaptive activation functions that have been shown to accelerate PINN training convergence, addressing some of the traditional problems of training deep learning models [6].

Nevertheless, this work aims to compare PINNs with ordinary NNs using numerically generated data from a simple, nonlinear, damped pendulum system and to analyze the results. Both networks will be trained on the full dataset, a linearly sparsified dataset, and a random uniformly sparsified dataset, and some features of this network will be discussed.

II. Mathematical Framework

A. Generating data using numerical methods

To investigate the predictive performances of both NNs and PINNs for a simple, nonlinear, and damped pendulum system, we must first introduce the dynamics of the nonlinear pendulum.

The simple, nonlinear, and damped pendulum was chosen because it is one of the simplest examples students are introduced to when learning about dynamical systems governed by nonlinear differential equations.

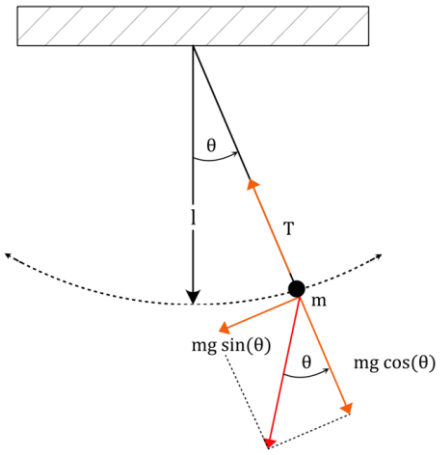

Fig. 1. Illustrative diagram of a pendulum system

In Fig. 1, we observe a pendulum system, where m is the pendulum's bob mass, l is its length, Θ is the angle of oscillation, and g is the acceleration due to gravity. From this information, we can derive the differential equation for the nonlinear motion of the pendulum by applying Newton's law along the tangent of the oscillation arc.

$$F = ma$$

$$ma = -mgsin(\theta)$$

$$\ddot{\theta} = -\frac{g}{L}sin(\theta) \quad (1)$$

The negative sign in Equation 1 indicates that the pendulum is decelerating as it moves toward the top of the arc. The only thing we are missing to complete the differential equation is the damping force, which can be modeled as follows:

$$\ddot{\theta} + \left(\frac{g}{l}\right) sin(\theta) + b\dot{\theta} = 0 \tag{2}$$

The damping force is modeled as a simple assumption, namely, that it is linearly proportional to the object's speed, with a constant of proportionality b, as shown in Equation 2.

Now that the physics model of the pendulum is derived, the final step involves the generation of numerical data using the Runge-Kutta 4 method [7].

1. Continuous-Time Model

$$\dot{\theta} = \omega$$

$$\dot{\omega} = -\left(\frac{g}{l}\right) sin(\theta) - b\omega \tag{3}$$

2. State-Space Representation

$$x_1 = \theta$$

$$x_2 = \dot{\theta}$$

$$\dot{x_1} = x_2$$

$$\dot{x_2} = -\left(\frac{g}{l}\right) sin(x_1) - bx_2 \tag{4}$$

3. Runge–Kutta 4 (RK4) Integration Scheme

$$\theta_{n+1} = \theta_n + \left(\frac{h}{6}\right)(k_1^1 + 2k_2^1 + 2k_3^1 + k_4^1)$$

$$\dot{\theta}_{n+1} = \dot{\theta}_n + \left(\frac{h}{6}\right)(k_1^2 + 2k_2^2 + 2k_3^2 + k_4^2) \tag{5}$$

In Equations 3, 4, and 5, the derivation of the integration model is shown, where k_1, k_2, k_3, and k_4 are intermediate slopes computed from the system dynamics.

The numerical solution of pendulum dynamics is shown in Fig. 2. The values of Θ across the whole time series were generated using Equation 5. The states were propagated forward in time with a fixed step size h, producing trajectories used as training data.

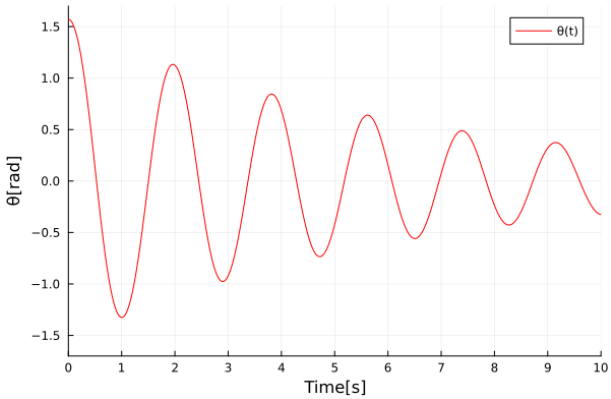

Fig. 2. Numerical genereted data using Eqations 5 (RK4) taking damping into consideration

TABLE I. CONSTANTS USED IN THE NUMERICAL GENERATION OF TRAINING DATA

Symbol	*Value*	*Description*
g	9.81 m/s^2	gravitational acceleration
l	0.7 m	pendulum length
b	0.3	damping constant
Δt	0.001	sample time
$\theta(t=0)$	1.57 rad	initial conditions
$\dot{\theta}(t=0)$	0 rad/s	
time span	0 – 10s	the time span during which the data was generated

B. Classical Neural Networks

NNs approximate unknown functions by composing affine transformations and nonlinear activation functions. Let $x \in R^n$ denote the input and $u(x,t)$ the target function. A neural network approximation can be written as

$$u(x,t) \approx h(x,t;\omega) \tag{6}$$

where h is a parameterized function defined by the network architecture, and ω denotes the trainable parameters. The main goal of NNs is to approximate the unknown function by adjusting the trainable parameters using an appropriate optimization algorithm.

For classical NNs, the optimization problem is purely data-driven; in our case, for the cost function, we use only the true and predicted data points to compute the loss, as shown in Equation 7.

$$\mathcal{L}_d = \frac{1}{N_d}\sum_{i=1}^{N_d} \|\theta_i - h(t_i;\omega)\|^2 \tag{7}$$

The main limitation of this type of network is that, for low quality or noisy data, it overfits and fails to achieve the expected approximation.

C. PINNs architecture

On the other hand, PINNs extend standard NNs by embedding physical laws, typically expressed as partial differential equations, into the learning process. In our case, the unknown angular displacement $\theta(t)$ is approximated by a neural network as shown in Equation 8.

$$\theta(t) \approx h(t;\omega) \tag{8}$$

For this to be PINNs, we need to embed physical laws; thus, we construct the physics-informed loss as the residual of the governing Equation 2.

$$\mathcal{R}(t) = \ddot{h}(t;\omega) + \left(\frac{g}{l}\right) sin(h(t;\omega)) + b\dot{h}(t;\omega) \tag{9}$$

The more general physics loss function is described in following Equation 10.

$$\mathcal{L}_p = \frac{1}{N_f}\sum_{j=1}^{N_f}\left\|\mathcal{R}(t_j;\omega)\right\|^2 \tag{10}$$

where N_f are collocation points, a set of points where the physics residual is computed; the most important thing is that these points do not require a known solution.

Temporal derivatives are computed via automatic differentiation. A special feature of PINNs is the avoidance of numerical discretization for calculating derivatives; instead, they are computed analytically via the chain rule. This ensures that the physics loss is not subject to truncation errors common in grid-based methods [8].

Finally, the total loss function of this network is shown in Equation (11).

$$\mathcal{L}_t = \frac{1}{N_d}\sum_{i=1}^{N_d}\|\theta_i - h(t_i;\omega)\|^2 + \frac{1}{N_f}\sum_{j=1}^{N_f}\left\|\mathcal{R}(t_j;\omega)\right\|^2$$
$$\mathcal{L}_t = \mathcal{L}_d + \mathcal{L}_p \tag{11}$$

where $\mathcal{L}_d$ is a data loss enforcing agreement with known solution values (e.g., initial and boundary conditions), and $\mathcal{L}_p$ is a physics loss penalizing violations of the governing ODE at collocation points within the domain. Both terms are typically formulated as mean squared errors. The network parameters are optimised by minimising a composite loss function that combines data, physics, and initial and boundary conditions.

This type of network can achieve good results even with sparse or low quality data – that is its advantage and will be discussed in the following text.

III. Training Setup

The main thing when using PINNs is the inclusion of a physics-informed component in the loss function based on the differential equation residual.

$$\mathcal{L}_d = \frac{1}{N_d}\sum_{i=1}^{N_d}\left|\theta_p - \theta_d^i\right|^2 \tag{12}$$

$$\mathcal{L}_p = \frac{1}{N_p}\sum_{i=1}^{Np}\left|\ddot{\theta}_f^i + \left(\frac{g}{l}\right)sin\left(\theta_f^i\right) + b\dot{\theta}_f^i\right|^2 \tag{13}$$

$$\mathcal{L}_i = \frac{1}{2}\left|\theta_p(t_0) - \theta_d(t_0)\right|^2 + \frac{1}{2}\left|\dot{\theta}_p(t_0) - \dot{\theta}_d(t_0)\right|^2 \tag{14}$$

$$\mathcal{L}_T = \lambda_d\mathcal{L}_d + \lambda_p\mathcal{L}_p + \lambda_i\mathcal{L}_i \tag{15}$$

As seen in Equation 13, the total loss $\mathcal{L}_T$ is calculated as the sum of the loss due to data $\mathcal{L}_d$, physics $\mathcal{L}_p$, and initial/boundary conditions $\mathcal{L}_i$, and is multiplied by configurable hyperparameters to have additional enforcement on the loss constraints λ_d, λ_p, and λ_i.

The neural network architecture consists of an input layer, three hidden layers with 32 neurons each, and a linear output layer. As shown in Table 2, the network employs a hybrid activation strategy: the first hidden layer uses a sin activation function, and the second and third layers use tanh activation functions. The choice of this arrangement is motivated by the use of various activation functions; optimal results were obtained with a network integrating sine and hyperbolic tangent functions, and this configuration better represents the periodic nature and oscillatory behavior [9].

Before the training part, the input data needed to be scaled using the Z-score method. This was crucial because PINNs can struggle when the input and output have very different scales [10].

The Adam optimizer (Adaptive Moment Estimation) was selected as the default optimization algorithm. For all tests, the step size was set to 0.001 and the number of epochs to 3000. The hyperparameters were set to λ_d=1, λ_p=0.1, λ_{ic}=0.05, and the number of collocation points for all experiments was fixed to N_f=100. For the training data, three configurations were evaluated: linearly spaced, uniformly distributed, and the complete dataset. In this way, the network's approximation performance was tested.

TABLE II. NEURAL NETWORK ARCHITECTURE

Layer	Type	Neurons	Activation
Input	Time(t)	1	/
Hidden 1	Fully connected	32	sin
Hidden 2	Fully connected	32	tanh
Hidden 3	Fully connected	32	tanh
Output	Angle(θ)	1	1

The code is implemented in Julia using the Lux library [11]. Training is run on a workstation with an Intel i5-12400F CPU (6 cores), 16 GB RAM, and an AMD Radeon RX 6600 GPU. The GPU is not utilized because the HIP SDK is not supported on Windows 10. Nevertheless, Lux.jl offers XLA (Accelerated Linear Algebra), which can speed up training even on a CPU.

IV. Results

As shown in Table 3, the tests were conducted with varying data sizes and two sparsification methods. In Fig. 3,

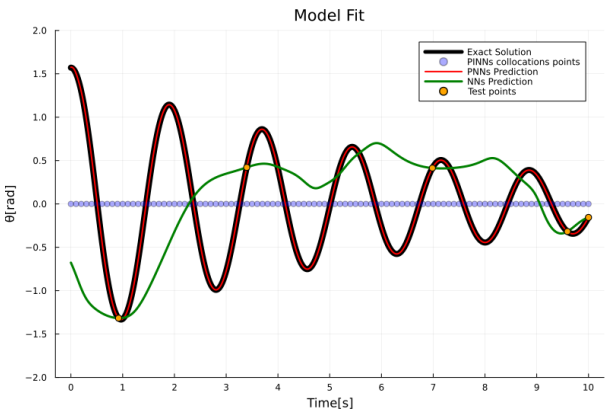

Fig. 3. Comparison of NN and PINN approximation based on 5 uniformly distributed training points.

the NN fails to predict the exact solution of the pendulum dynamics with uniformly distributed 5 data test points.

TABLE III. RMSE VALUES FOR VARIATIONS OF NP

TABLE IV. POINTS OF VARIABLY-SPACED TRAINING POINTS

	Uniformly-distributed data points		Linearly-spaced data points	
Np	NN	PINN	NN	PINN
5	1.4217	0.0015	1.3564	0.0069
25	0.2643	0.0084	0.2437	0.0004
50	0.1121	0.0003	0.0071	0.0004

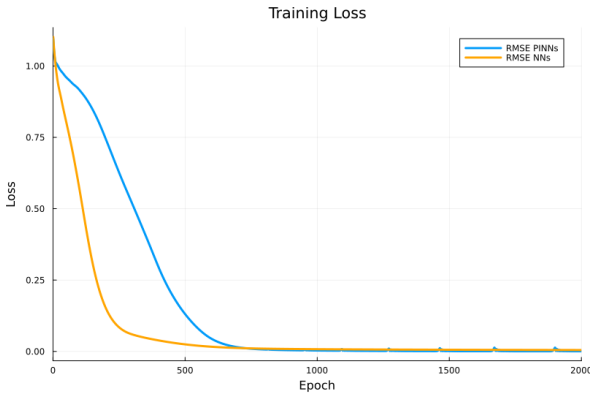

Fig. 4. Comparison of PINNs and NNs tested with 100 linearly spaced training points. NN converges to an accurate solution after 240 iterations, while PINNs solution faces a delayed convergence after 600 iterations.

The inability of NNs to approximate is first caused by a lack of data and, second, by uniform sparsification, which reduces the accuracy of NNs because these samples fail to capture the local curvature and phase of the nonlinear pendulum dynamics, causing the network to interpolate across large temporal gaps.

In Table 3, PINNs outperform NNs with fewer data points. With linearly spaced data points, NNs' RMSE remains fairly accurate with 50 data points; with fewer than that, performance deteriorates significantly. Even with 50 data points, NN performance is inadequate for uniformly distributed data. This is because NNs rely heavily on available data, and when the data are insufficient, they tend to overfit. In comparison, PINNs can still approximate the solution with less data, achieving much lower generalization loss because they have an embedded physical loss that drives the solution toward the exact solution.

Figure 4 shows that the NN converges faster than the PINNs. This is because NNs solve a simpler, unconstrained interpolation problem, and by doing so, they rapidly converge to fit the given data points. However, PINNs have to do more iterations as they explore a complex multi objective landscape that has to compromise the data accuracy with the physical ODE residuals.

V. CONCLUSION

As seen from the results, PINNs outperform classical NNs in a sparse-data setting; on the other hand, if there is enough data for training, the recommended network should be NNs, because they can converge rapidly with sufficient data points and require less computational power. In most cases, PINNs are structured with complex architectures and complicated loss functions. These loss functions are filled with a lot of local minima, and global minima cannot always be guaranteed. Even a slight change in initial weights and distribution of data points can change the stability and robustness. Hence, more robust NN architectures are required to overcome this problem.

To gather more information, the goals of future work should incorporate noisy data and data from a real pendulum system, and other architectures should be tested, for example, inverse PINNs (iPINNs).

ACKNOWLEDGMENT

This research has been supported by the research grants of the Serbian Ministry of Science, Technological Development, and Innovations, grant No. 451-03-33/2026-03/ 200066.

REFERENCES

[1] Raissi, M., Perdikaris, P., and Karniadakis, G. E., “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, Feb. 2019, doi: 10.1016/j.jcp.2018.10.045.

[2] B. Moseley, A. Markham, and T. Nissen-Meyer, “Solving the wave equation with physics-informed deep learning,” arXiv preprint arXiv:2006.11894, 2020.

[3] M. Raissi, A. Yazdani, and G. E. Karniadakis, “Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations,” Science, vol. 367, no. 6481, pp. 1026–1030, 2020.

[4] F. Sahli Costabal, Y. Yang, P. Perdikaris, D. E. Hurtado, and E. Kuhl, “Physics-informed neural networks for cardiac activation mapping,” Frontiers in Physics, vol. 8, art. no. 42, 2020.

[5] Sa, J., Jeon, B., Seo, Y., et al., “Physics-informed neural networks for complex fluids: opportunities and limitations,” Korea-Australia Rheology Journal, vol. 37, pp. 469–492, 2025, doi: 10.1007/s13367-025-00140-6.

[6] A. D. Jagtap, E. Kharazmi, and G. E. Karniadakis, “Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems,” Computer Methods in Applied Mechanics and Engineering, vol. 365, 15 Jun. 2020, Art. 113028, doi: 10.1016/j.cma.2020.113028.

[7] E. Hairer, S. P. Nørsett, and G. Wanner, Solving Ordinary Differential Equations I: Nonstiff Problems, 2nd rev. ed. Berlin, Germany: Springer-Verlag, 1993, doi: 10.1007/978-3-540-78862-1.

[8] T. G. Grossmann, U. J. Komorowska, J. Latz, and C.-B. Schönlieb, "Can physics-informed neural networks beat the finite element method?" *IMA Journal of Applied Mathematics*, vol. 89, no. 1, pp. 143–174, Jan. 2024, doi: 10.1093/imamat/hxae011.

[9] M. Byra, C. Poon, M. F. Rachmadi, M. Schlachter, and H. Skibbe, "Exploring the performance of implicit neural representations for brain image registration," *Scientific Reports*, vol. 13, no. 1, p. 17334, 2023, doi: 10.1038/s41598-023-44517-5.

[10] X. Chen, Y. Wang, Q. Zeng, X. Ren, and Y. Li, “A two-step scaled physics-informed neural network for non-destructive testing of hull rib damage,” *Ocean Engineering*, vol. 319, p. 120260, Mar. 2025, doi: 10.1016/j.oceaneng.2024.120260.

[11] A. Pal, Lux: Explicit Parameterization of Deep Neural Networks in Julia, version 0.5.0. Zenodo, Apr. 2023. doi: 10.5281/zenodo.7808904. [Online]. Available: https://lux.csail.mit.edu/.