Solving real-world optimization tasks using physics-informed neural computing

Manan Lohani · Follow

12 min read · Just now

This reproducibility project was conducted by the students from TU Delft as part of the CS4240 Deep Learning course (Q3 2024).

Akansha Mukherjee | 5973767 | a.mukherjee-11@student.tudelft.nl

- Swinging up of pendulum reproduction
- Swinging up of pendulum replication using JAX

Bakul Jangley | 6055826 | bjangley@student.tudelft.nl

- Fermat principle reproduction
- Ablation study

Karthik B Nair | 5967333 | k.bijunair@student.tudelft.nl

- Spacecraft trajectory reproduction
- New data (to check robustness of the PINN)
- Swinging up of pendulum replication using JAX

Manan Lohani | 5915821 | m.j.lohani@student.tudelft.nl

- Fermat principle replication using JAX
- Hyperparameters check and study

The aim of this project was to reproduce the results obtained in the paper “Solving real-world optimization tasks using physics-informed neural computing” by Seo,J based on the source code provided by the authors.

Links to the original paper and source code:

- Original Paper
- Source Code

1.Introduction

What are PINNs?

Physics-informed neural networks (PINNs) are a machine learning-based optimization scheme that incorporates physics with the operational objectives. They solve differential equations using neural networks, thereby learning the underlying physics of a given problem.

The governing physics laws that dictate the evolution of a system are introduced as an additional objective function along with sparse (or absent) data in the form of boundary conditions while training the neural network. In addition, PINNs can also integrate a third objective function that guides the system towards the goal.

The function that the neural network is expected to represent is a mapping from the independent variable(s) within a given domain to the optimal path of “design” variables. That is, the neural network takes domain variables as inputs and outputs the design variables. The objective function to minimise is a weighted combination of three loss components as shown.

$$L_{phys}(\theta) = \frac{1}{N_\Omega}\sum_{j=1}^{N_\Omega} \|\mathcal{F}(t_j;\theta)\|_2^2 \text{ for } t_j \in \Omega.$$

$$L_{con}(\theta) = \begin{cases} \frac{1}{N_{\partial\Omega}}\sum_{j=1}^{N_{\partial\Omega}} \|\hat{u}(t_j;\theta) - BC\|_2^2 \text{ (Dirichlet)} \\ \frac{1}{N_{\partial\Omega}}\sum_{j=1}^{N_{\partial\Omega}} \|\frac{d\hat{u}}{dt}(t_j;\theta) - BC\|_2^2 \text{ (Neumann)} \end{cases} \text{ for } t_j \in \partial\Omega.$$

Figure 1: Physics loss and Constraint loss formulae

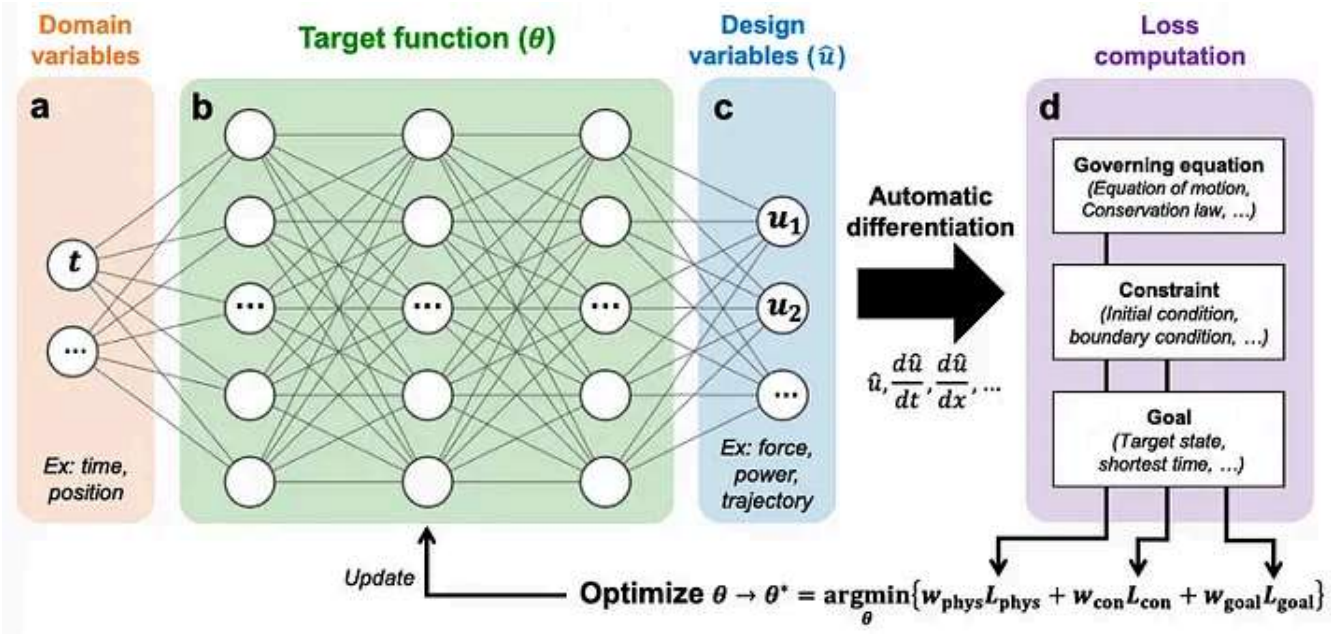


Figure 2: Physics Inspired Neural Network Architecture

The main difference of this approach with traditional PINN solving differential equations is the objective function for the goal state. This component makes it applicable to real-world optimization problems such as finding an optimal path, etc.

The authors looked at four main applications and we introduce the core concept along with the underlying loss functions for these below.

Swinging up of a pendulum

Swinging up a fixed-axis pendulum is a popular problem for testing the performance of a controller. While classical algorithms such as PID controllers work well for this task, they fail in scenarios where the maximum available torque is limited, making it insufficient to invert the pendulum in one go. Greedy controllers, being short sighted, would also fail in such cases. In such scenarios, the controller is expected to swing the pendulum to and fro in order to gain enough momentum to invert it to a vertically upright position.

The domain variable time goes from 0 to 10 seconds while the design variables are the angle (ϕ) and unnormalized torque (τ). The physics loss is computed from the equation:

$$\mathcal{F} = ml^2\ddot{\phi} - (\tau - mgl\sin\phi).$$

Two initial conditions are specified for the angle and torque at t = 0. A third Neumann initial condition is specified for the time derivative of the angle (angular

velocity). The target state at t=10s is incorporated in the form of a custom boundary condition. The goal is to reach a vertically inverted state at t = 10s

Shortest Distance Path

In this problem, we optimize the time taken or cost to reach a 'goal' in situations where an analytical solution has been derived. The difference between this problem and the inverted pendulum is that time is taken as a variable and minimized.

The Fermat's Principle states the path followed by a ray of right between two points is the shortest time path between those two points. It accounts for the change in path when the refractive index of the medium changes. The loss function chosen the implementation of PINN is

$$\mathcal{F} = \left(\frac{1}{T}\frac{dx}{dt_N}\right)^2 + \left(\frac{1}{T}\frac{dy}{dt_N}\right)^2 - \left(\frac{c}{n}\right)^2$$

Where c is the speed of light, T is the time taken from the start to the end and the boundary conditions include the start and goal positions.

The Brachistochrone curve represents the path of shortest descent time for an object moving under the force of gravity without friction. This problem was first posed by Johann Bernoulli in 1696, and the solution draws upon the calculus of variations. The loss function used is based on mechanical energy conservation:

$$\mathcal{F} = gy_0 - \left(gy + \frac{1}{2}\left(\left(\frac{1}{T}\frac{dx}{dt_N}\right)^2 + \left(\frac{1}{T}\frac{dy}{dt_N}\right)^2\right)\right)$$

Swing-by Trajectory of a Spacecraft

In the scenario described, the spacecraft aims to reach its destination using minimal fuel by exploiting gravitational forces in a dynamic environment with multiple astronomical objects. The objective is to determine a trajectory that enables the spacecraft to achieve its target position without relying on onboard propulsion. For this task, the physics loss is computed as the net thrust F, as indicated by the equation below:

$$\mathcal{F} = \text{Thrust vector} = \begin{cases} \frac{1}{T^2}\frac{d^2x}{dt_N^2} - \sum_{(x_o,y_o,GM_o)} \left(\frac{-GM_o(x-x_o)}{\left((x-x_o)^2+(y-y_o)^2\right)^{1.5}} \right) & \text{(x component of thrust)} \\ \frac{1}{T^2}\frac{d^2y}{dt_N^2} - \sum_{(x_o,y_o,GM_o)} \left(\frac{-GM_o(y-y_o)}{\left((x-x_o)^2+(y-y_o)^2\right)^{1.5}} \right) & \text{(y component of thrust)} \end{cases}$$

Constraints are set as the coordinates of the starting and ending points, respectively (−1, −1), (1, 1).

2.Reproducibility Project

For our project we focused on the following tasks to evaluate and understand the original code provided by the authors:

- **Reproduction:** We reproduced the code provided by the author in his github page for all the tasks using Google Colab and evaluated the results obtained.
- **Replication:** We developed a complete implementation from scratch for the pendulum and Snell's principle using JAX instead of Tensorflow. The original code relies of deepxde which was not used by us in our reproduction. JAX's jitted functions were used which ensured faster processing and improved computation.
- **Ablation Study:** We undertook a study to understand the output of the models when removing or adding hidden layers for the tasks. This helps to demonstrate the importance of the selected architecture.
- **Hyperparameters check:** We fine tuned the selected hyperparameters like learning rate, loss weights, number of epochs for all four models and evaluated the results to understand the difference in the output when these hyperparameters are tweaked.
- **New Data**
  Since PINN models do not require a traditional dataset since they incorporate physics laws to generate data points, we opted to test the model robustness instead. This was carried out by increasing the physics complexity (adding friction), and injecting gaussian noise.

We will briefly discuss the work done in each of these steps and the corresponding results obtained below.

3.Reproduction

We reproduced the code provided by the author using Google Colab and successfully achieved similar results to those presented in the paper. The author developed four models for four tasks: swinging up a pendulum, Snell's principle, Brachistrone curve, and spacecraft trajectory. We recreated the plots obtained for each of these tasks and comprehended the underlying mechanisms involved in the original code.

4. Replication

We faced several challenges during the replication process, primarily stemming from poorly documented source code, which we highlight below:

- *deepxde* allowed for an easy formulation of differential equations of the physics models, and adding external trainable parameters (besides the model parameters). The latter is an important feature in two tasks that the PINN is trained on, as the **total time duration is a trainable parameter** that needs to be updated every step, through back-propagation. Although the rest of the code could be written, adding time as a trainable parameter was not straightforward to implement using flax, given the time constraints.
- Relative weights of the losses: the paper specified the relative weights for the physics loss, constraint loss and goal loss as (1,10,1). There were three components in the constraint loss, and it was not obvious if all three of them were weighted 10 or just one.
- Dataset generation: The size and nature of the dataset was not specified by the authors.
- Training method: the variant of gradient descent used (SGD, batch gradient descent, etc.) was not obvious from the literature.
- Neumann Boundary Conditions: Some background knowledge on the physical and mathematical implications of Neumann Boundary Conditions would have been helpful.

To replicate these models, we instead focused on using the JAX framework as it has recently been popularised for complex networks specifically for its faster computational speed. We attempted to develop the loss functions and the network

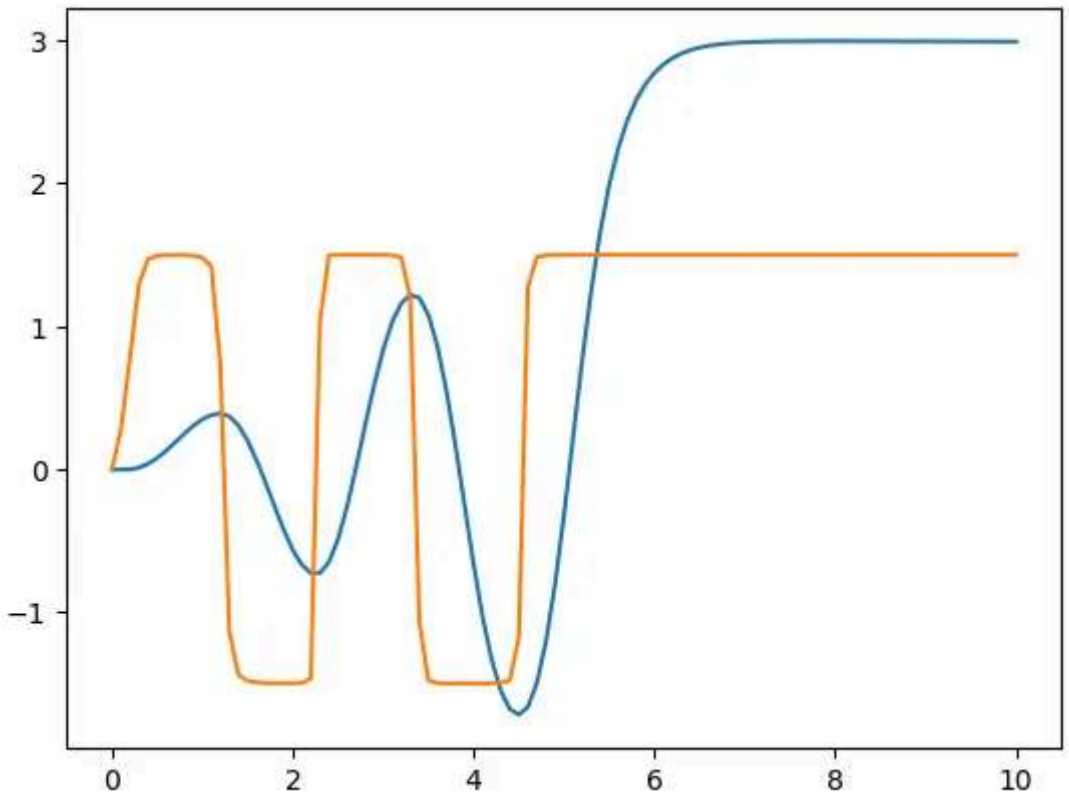
from scratch and more importantly without the help of any external libraries like deepxde. This resulted in a challenging and uphill task but we were able to replicate the pendulum model.

We explain our methodology and the results obtained below:

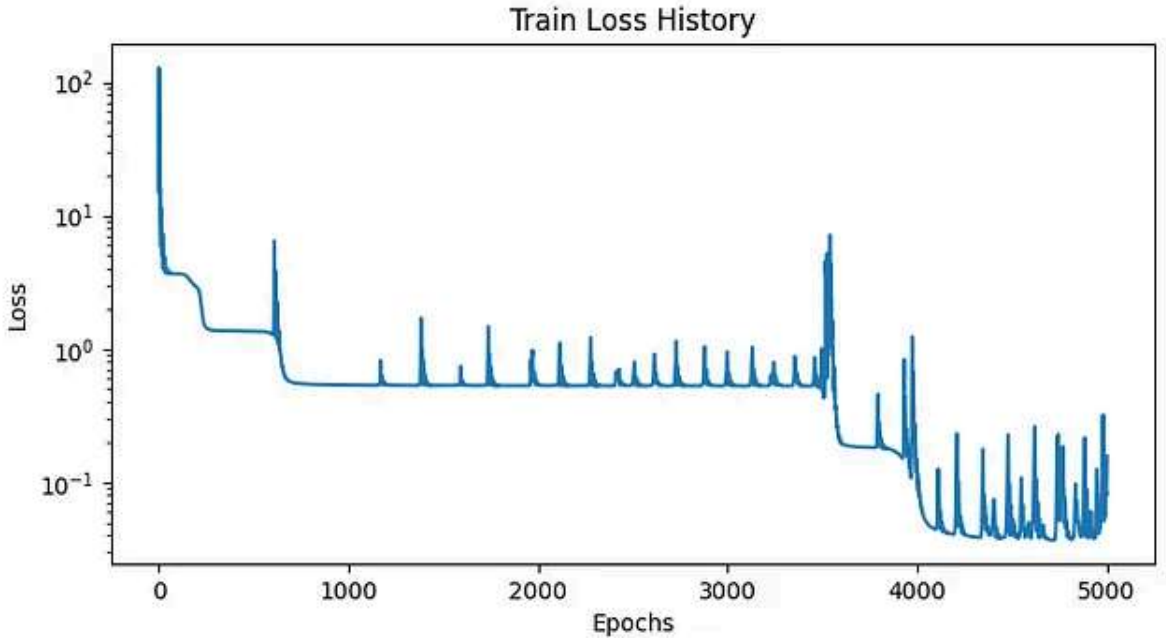
Swinging up a Pendulum

It was successfully reproduced on the jax framework, drawing inspiration from the repository provided by the authors. The following modifications were made to obtain the desired result. The remaining aspects of the implementation were preserved exactly:

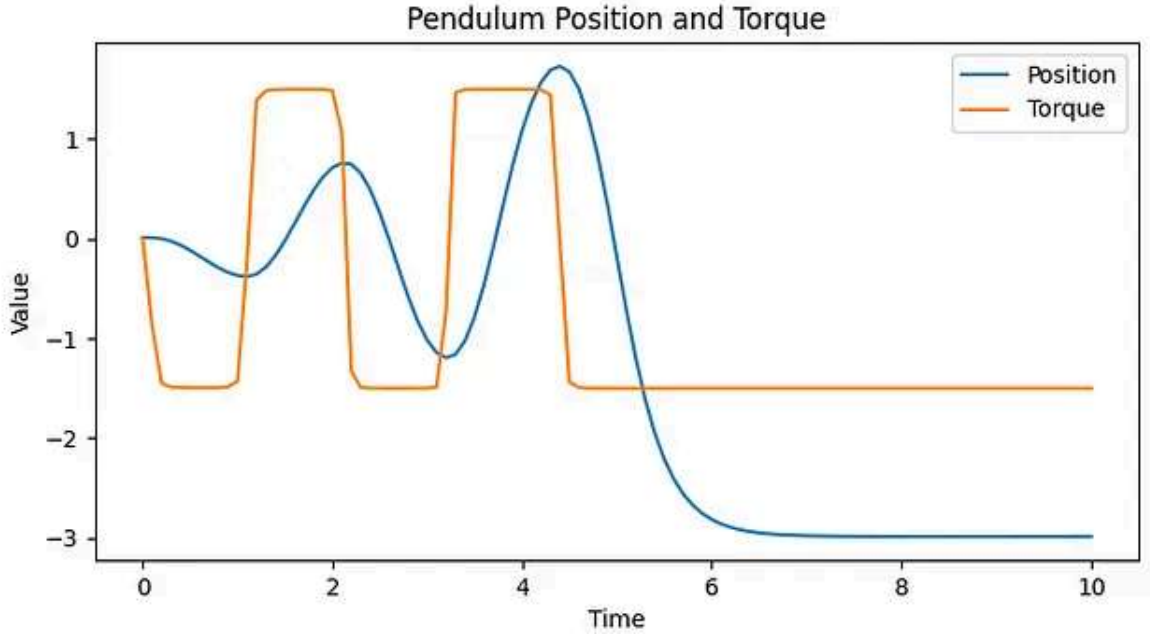
- The original learning rate of 2e-2 was found to be too high for the reimplementation. The loss was too high and the evolution of the design variables failed to capture the underlying physics of the problem. A learning rate of 1.5e-2 produced the same results as the original implementation.
- The authors of the original paper have used 5000 training iterations on Adam followed by a second optimization phase with LBFGS. This was omitted in the reimplementation and the performance was compared to the original implementation for the first optimisation phase only.



Author Implementation for Pendulum: Best loss: 3.99e-02



Training Loss for our Implementation



Output for our Implementation:Best loss: 0.0368

Differences: Due to random initialization and differences in internal implementations of the library used by them (deepxde) and us (jax, flax).

Inverted plots: At the vertically upright position, $\cos(\phi) = -1$ or $\phi = \pi / -\pi$. One of the plots reaches the goal position clockwise and the other reaches it counterclockwise.

5.Hyperparameter Tuning:

We noticed that the author designed models had various hyperparameters and it was not clearly mentioned in the paper about the reasoning behind their value

selection and the impact of this on the final output. The main hyperparameters that we selected for further understanding were: Learning rate, number of input samples, number of epochs for Adam optimiser and also the loss weights. We manually tweaked each of these individually while keeping the other hyperparameters fixed.

It is to be noted that to observe a trend, we selected a small set of hyperparameter values with high variance among them. This means that if the author selected a learning rate of 0.001, we used [0.01, 0.001,0.0001] to observe how the output varies.

Swinging up of a pendulum:

It was observed that for a pendulum model all the hyperparameters played a significant role on the final output. As compared to the other physics systems, this pendulum model was most likely to fail if the hyperparameter values were chosen incorrectly. The table below summarizes the results.

Hyperparameters	**Swinging up a pendulum**
Learning Rate	LR of 0.002 did not converge 0.02 and 0.2 converge
Input Size	2000 input points worked perfectly. Smaller values failed
Adam Steps	5000 epochs converges. 3000 and 6000 epochs failed
Physics Loss Weight	Weight of 1 converges to the ideal path. 10 and 0.1 did not work
Goal Loss Weight	Weight of 1 converges to the ideal path. 10 and 0.1 did not work
Boundary Loss Weight	Weight of 10 converges to the ideal path. 1 and 0.1 did not work

Shortest Time Path

It could be seen that for Fermat's principle model, the critical hyperparameters that had a major impact on the final output were Learning rate, input size and the weight losses. Adam epochs was not found to be a critical hyperparameter. On the other hand, for the Brachistochrone model, learning rate, Adam epochs and weight losses

are of importance. The input size did not seem to play a significant role for this model.

Hyperparameters	Fermat's Principle	Brachistochrone
Learning Rate	LR of 0.01 did not converge LR<=0.001 converge to endpoint	LR of 0.01 did not converge LR<=0.001 converge to endpoint
Input Size	Input size 1000 works perfectly. Other values did not converge	Values from 100 to 2000 converged to endpoint
Adam Epochs	Values greater than 1000 converged	3000 did not converge.
Physics Loss Weight	Weight of 10 did not converge 1 and 0.1 converged	Weight of 10 did not follow ideal path 1 and 0.1 converged
Goal Loss Weight	0.1 did not converge 0.01 and 0.001 converged	0.1 did not converge 0.01 works perfectly
Boundary Loss Weight	Smaller weight of 0.1 did not converge	Smaller weight of 0.1 did not converge

Swingby of Spacecraft

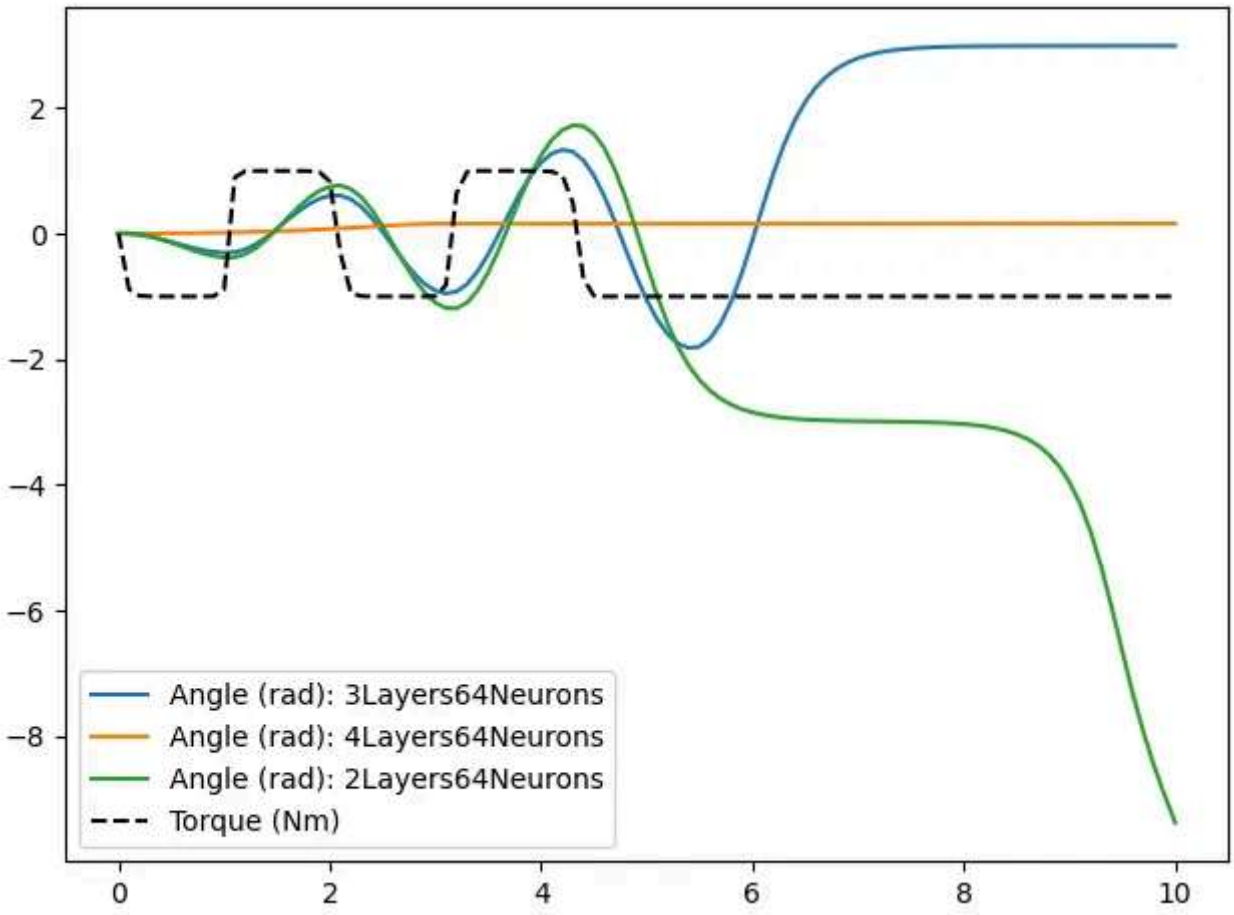
We observed that for this model, the most critical hyperparameters are learning rate, input size and boundary loss weight. The other terms did not cause a significant variation in the final output.

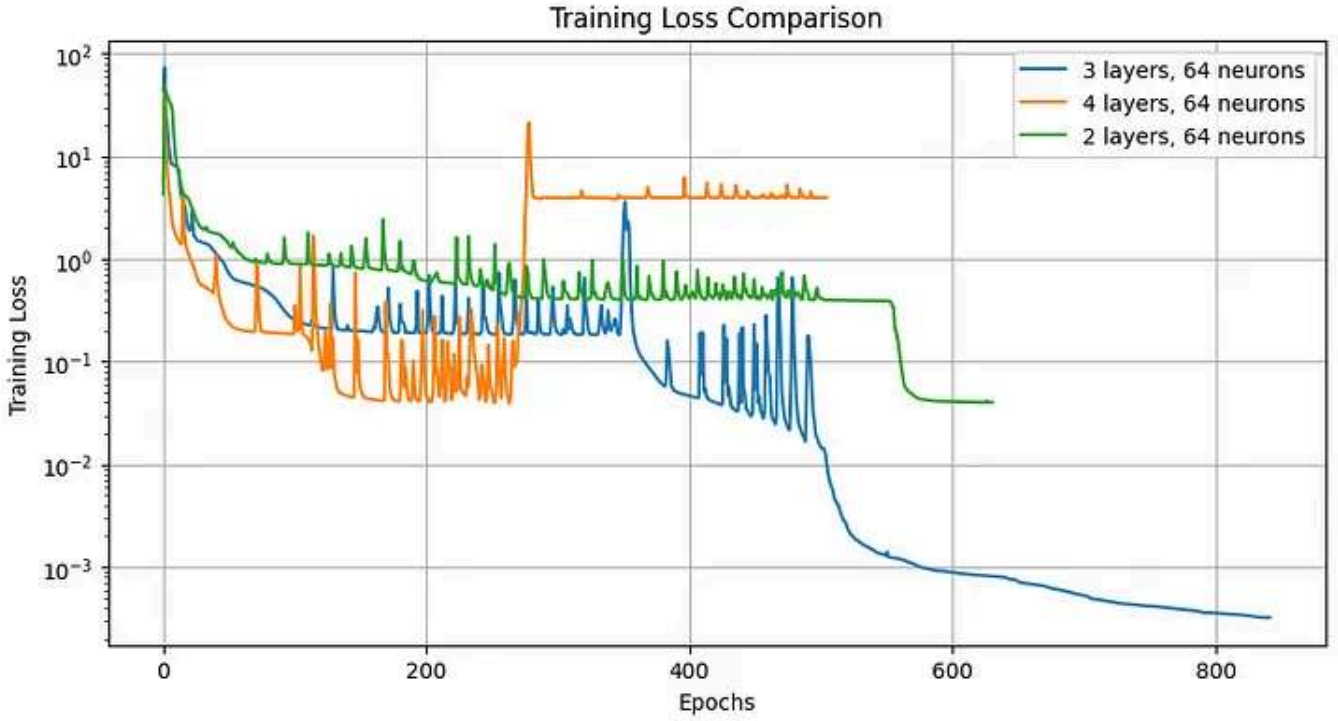
Hyperparameters	Swingby of Spacecraft
Learning Rate	Higher LR of 0.01 did not work 0.001 and lower converged
Input Size	Small input size of 100 failed 1000 or more values converged to endpoint
Adam Epochs	1000 or more epochs worked perfectly
Physics Loss Weight	No change observed with varying from 0.1 to 10
Goal Loss Weight	No change observed with varying from 0.1 to 10
Boundary Loss Weight	0.1 did not converge to the ideal path. 1 and 10 converged

6.Ablation Study

In the ablation study for implementing the PINN architecture described by the authors, we assess the impact of varying neural network depths on the model's performance. We use the model's predictions for different configurations: a network with two layers of 64 neurons, three layers of 64 neurons, and four layers of 64 neurons, alongside an analytic solution.

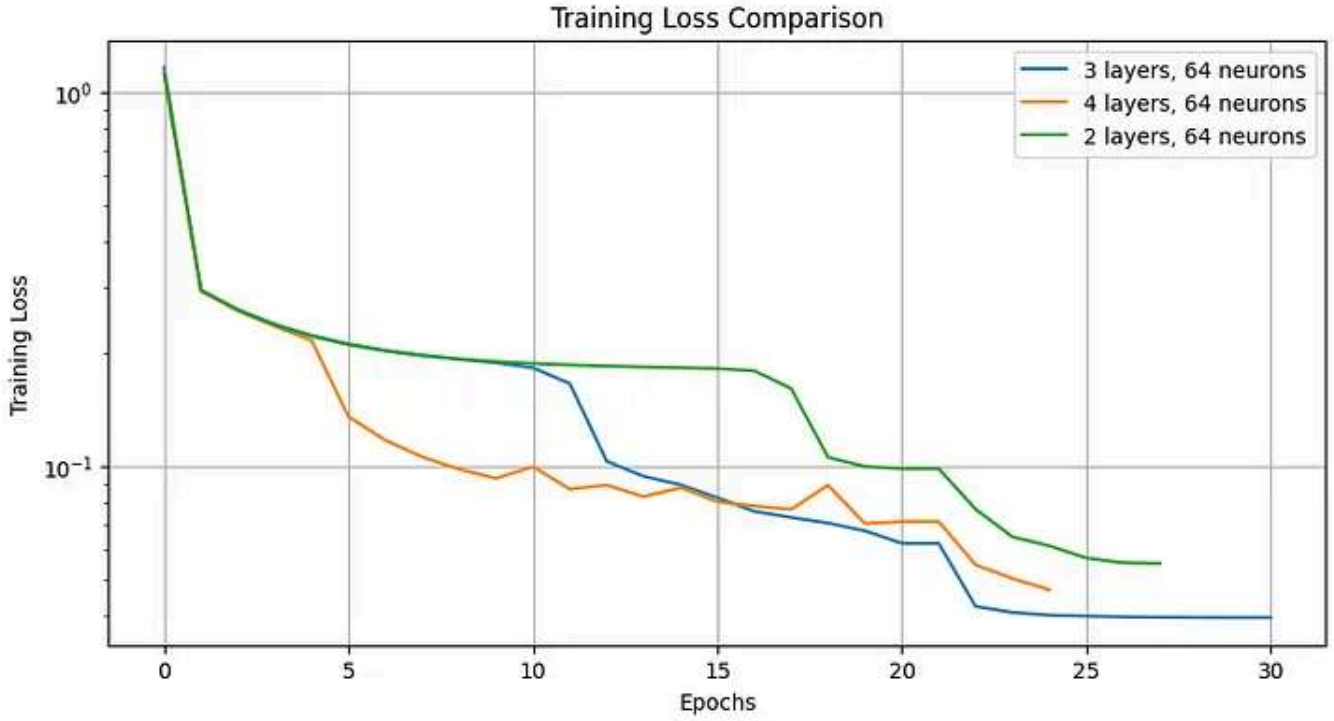
Inverted Pendulum:

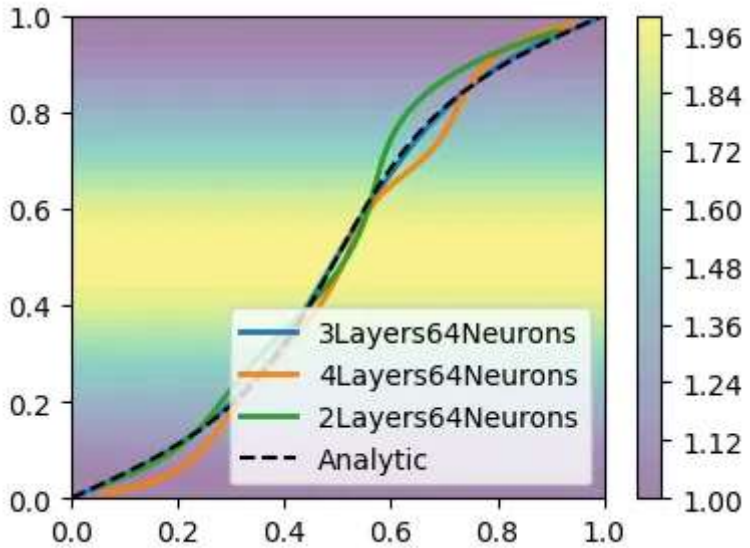




The training loss comparison plot explains the differences in convergence behaviors among the different neural network architectures. The two-layer network, while achieving a lower loss more rapidly, plateaus early, which may indicate an inability to capture more complex dynamics of the system. The four-layer network, despite showing a reduction in loss, shoots up at some point. The three-layer network, on the other hand, continues to demonstrate a gradual but consistent decrease in training loss over a large number of epochs, highlighting its capacity for effectively learning the system dynamics.

Fermat's Principle:



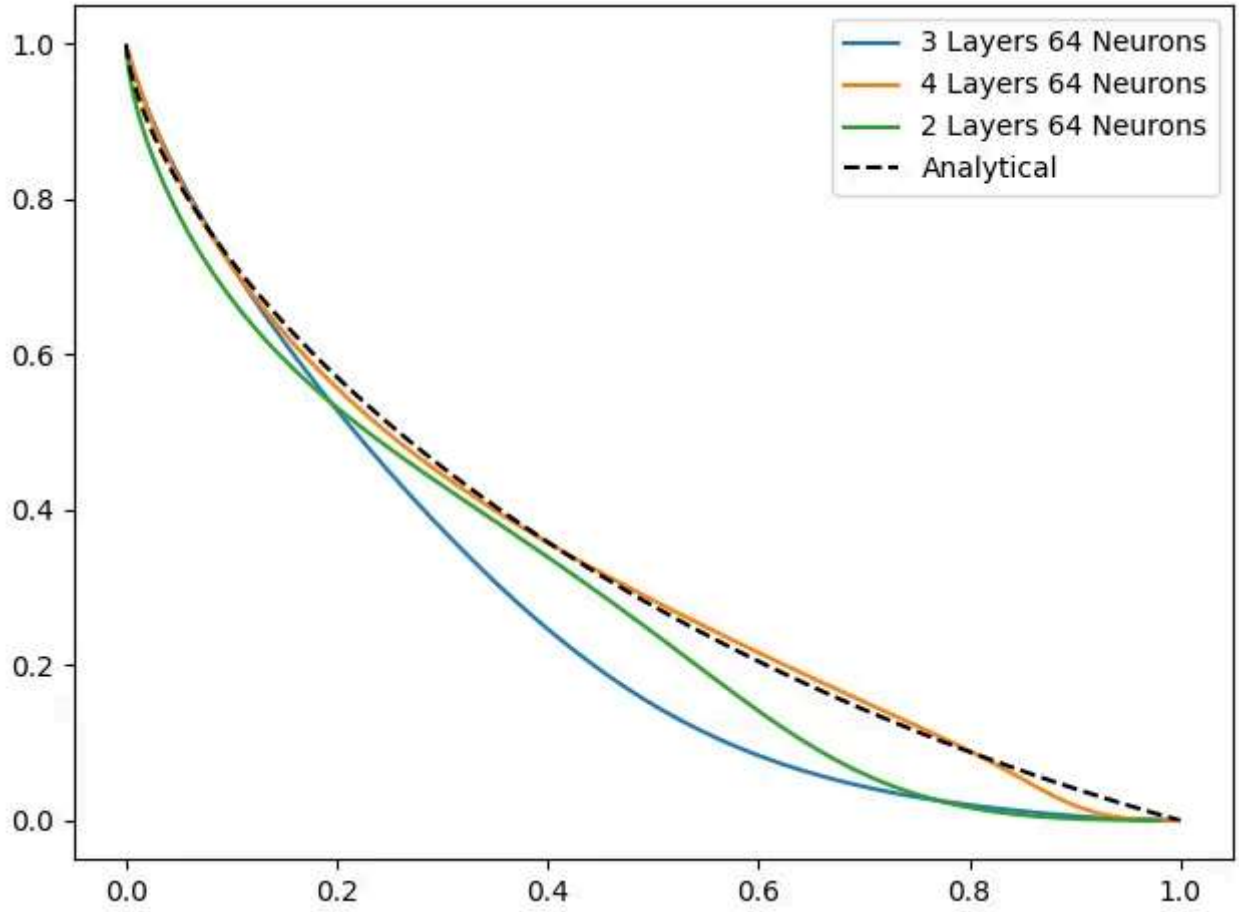


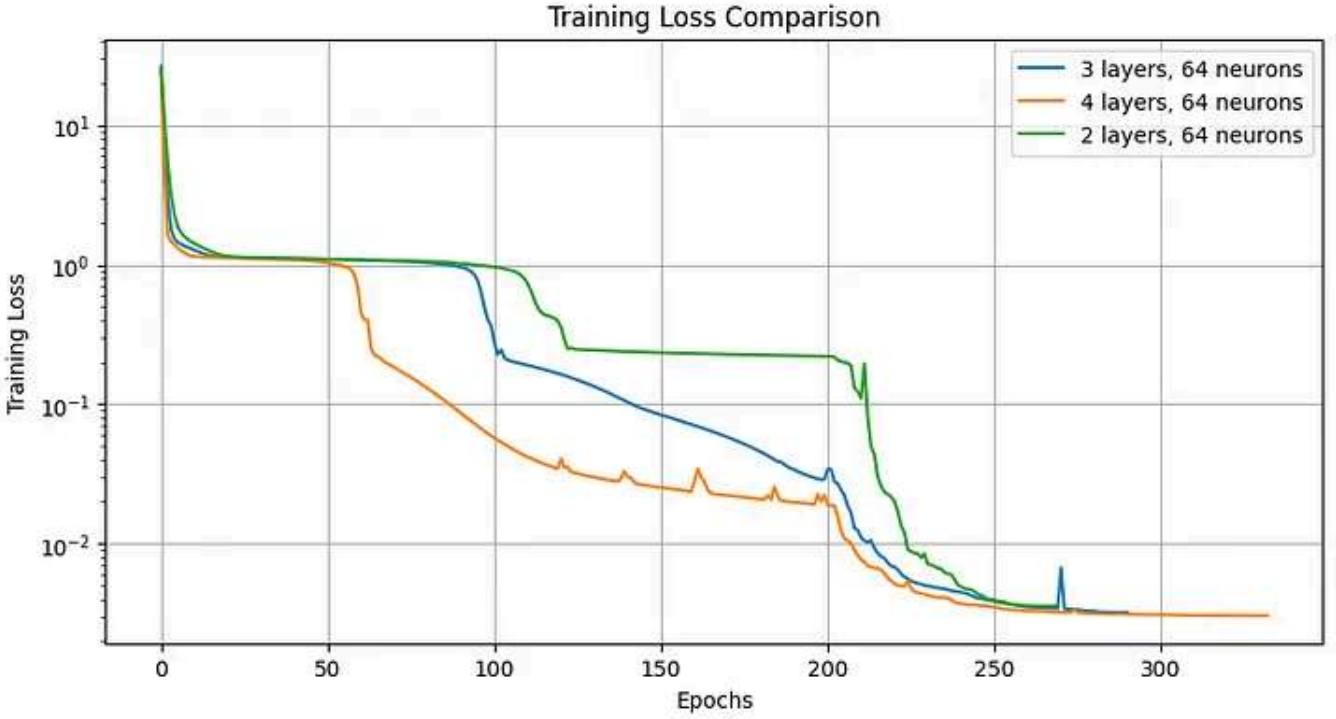
The three-layer model's solution closely aligns with the analytic curve, suggesting an optimal balance between model complexity and the ability to generalize the underlying physical law described by Fermat's principle. The four-layer and two-layer models exhibit slight deviations from the analytic solution.

Brachistochrone Curve:

The four-layer network exhibits an initial steep decrease in training loss, followed by a stabilisation, while the three-layer network shows a more gradual but continuous decline, indicating a steady learning process. The two-layer network, despite an initial rapid descent, reaches a plateau earlier than the others, suggesting limited learning capability beyond a certain complexity level. All three networks

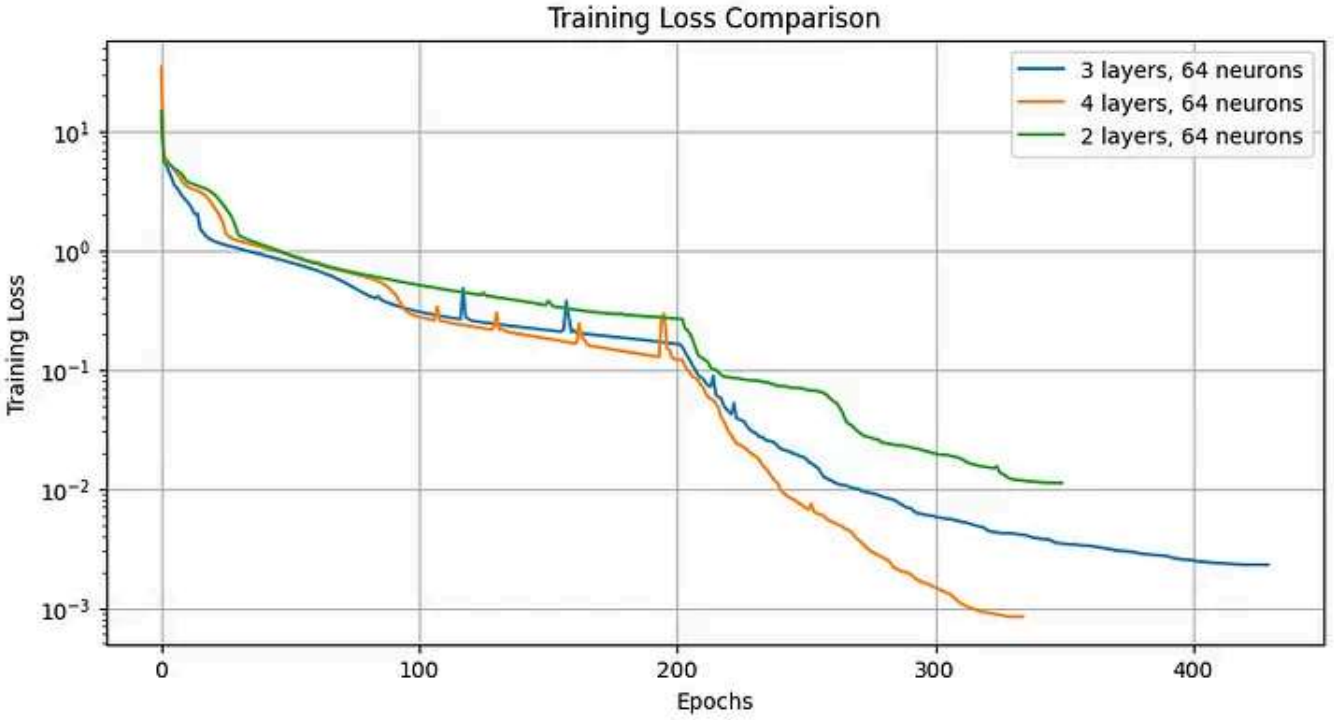
appear to follow the analytical solution, with the four-layer network nearly coinciding with the analytical curve throughout the domain.

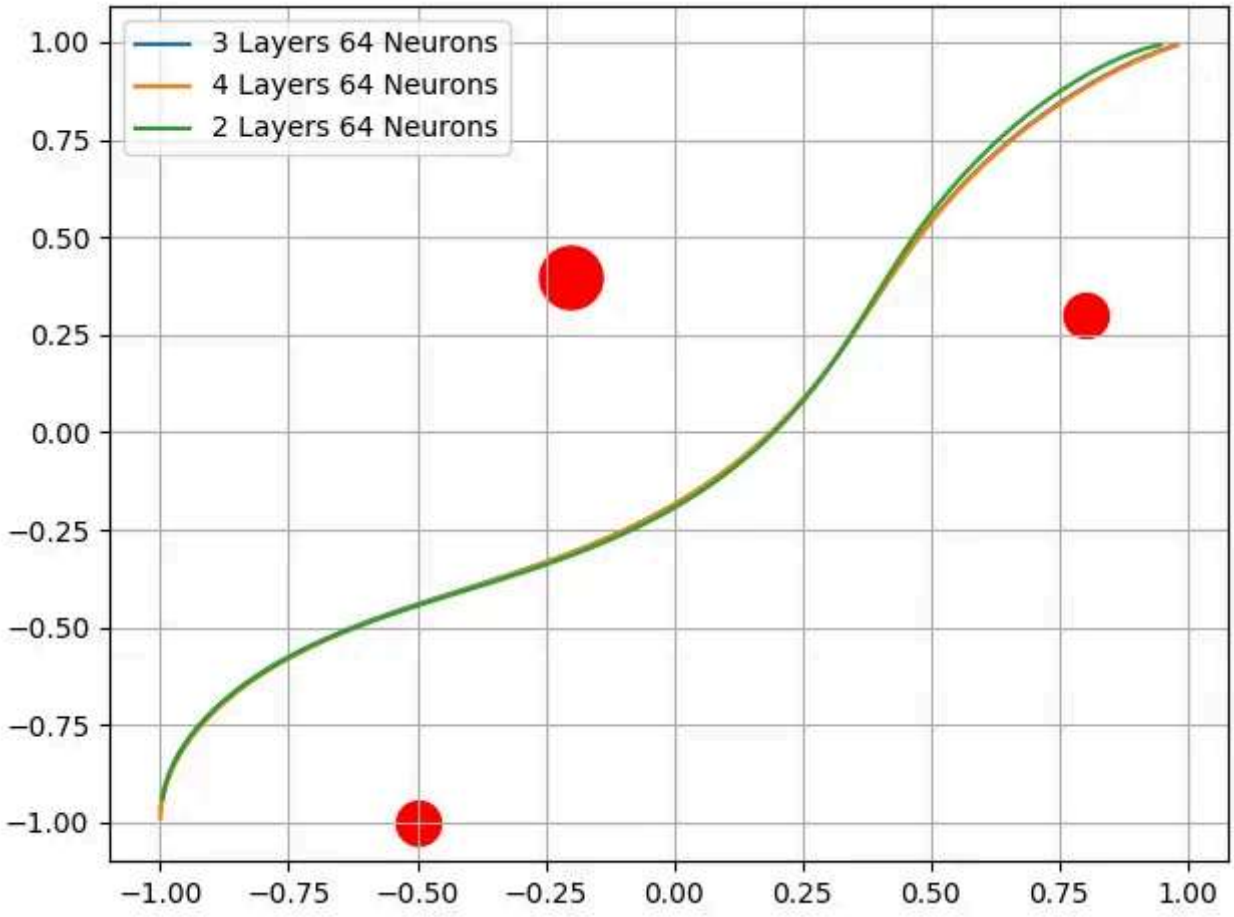




Spacecraft Trajectory:

The three-layer configuration achieves a consistent and smooth reduction in loss, outperforming the two-layer network, which plateaus earlier, and the four-layer network, which shows a less stable loss descent with possible overfitting tendencies. Notably, the two-layer network maintains a higher loss throughout training, indicating potential underfitting due to insufficient model complexity.





7. New Data

The next objective was to test the robustness of PINNs in learning complex strategies on physical systems, extending beyond the specific scenarios proposed by the authors. The first methodology adopted was to subject the PINN to a more complex physical model, by adding resistive, frictional torques to the pendulum.

$$\tau = m \cdot l^2 \cdot \ddot{\theta} + m \cdot g \cdot l \cdot sin(\theta) + b \cdot \dot{\theta}$$

Where b is the friction coefficient, ranging from 0.01 to 0.2. Subsequently, the model was trained using the updated physics equations. The result was that the model succeeded in adapting to a more complex physical phenomenon, without the need for hyperparameter tuning. An interesting outcome of the experiment was that the model was able to come up with new strategies for inverting the pendulum for various values of friction. Figures (a) and (b) illustrate the new behaviour learnt. The figure indicates that for higher friction values, the pendulum needs to oscillate more to reach the top, which aligns with intuition. The next test was to inject gaussian noise to the differential equation of pendulum physics. This test mimics real-world scenarios where noisy sensor data is encountered, leading to a more

realistic assessment of the model's performance. The noise was added to the updated torque equations with friction already active (b = 0.15). The mean was set to zero and the variance at 1% (of the torque estimated). The updated dynamics formulation now looks like:

$$\tau_{noisy} = \tau \cdot (1 + \epsilon), \quad \epsilon \sim N(\mu, \sigma^2)$$

Interestingly enough, without the need to tune hyperparameters, the model learnt the same behaviour, indicating robustness to noise as well as increasing physics complexity. Figures b and c indicate this similarity:

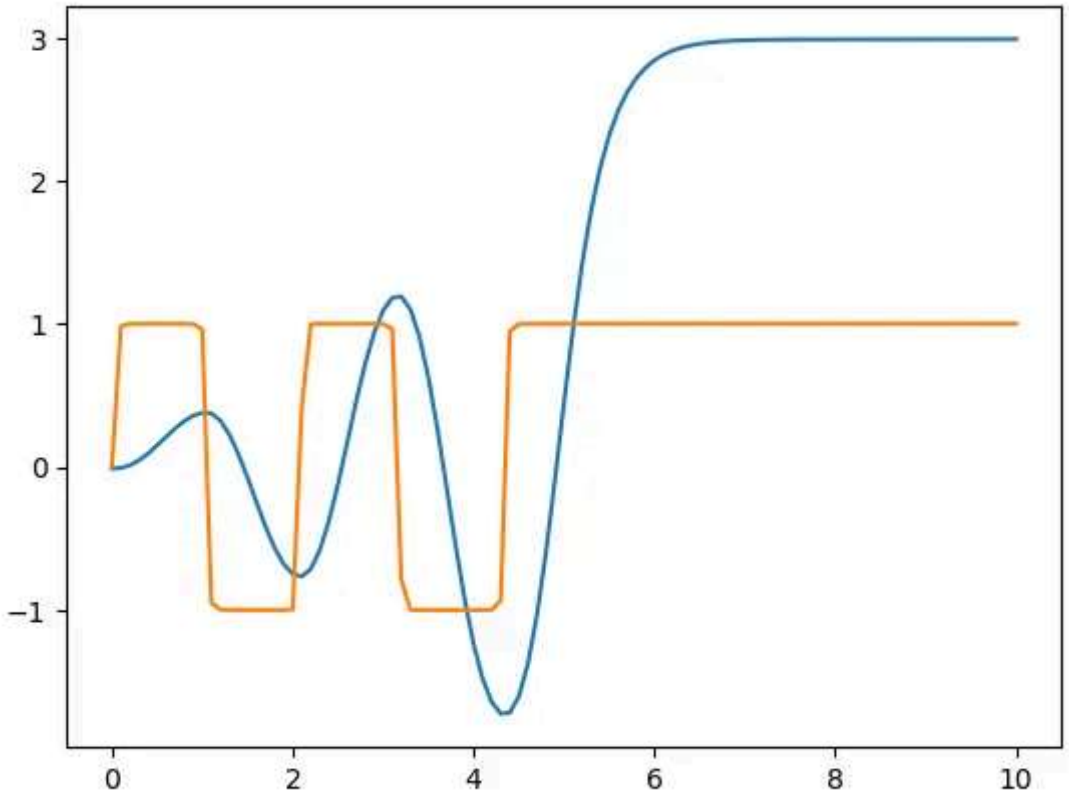


Fig: a) Output for zero friction

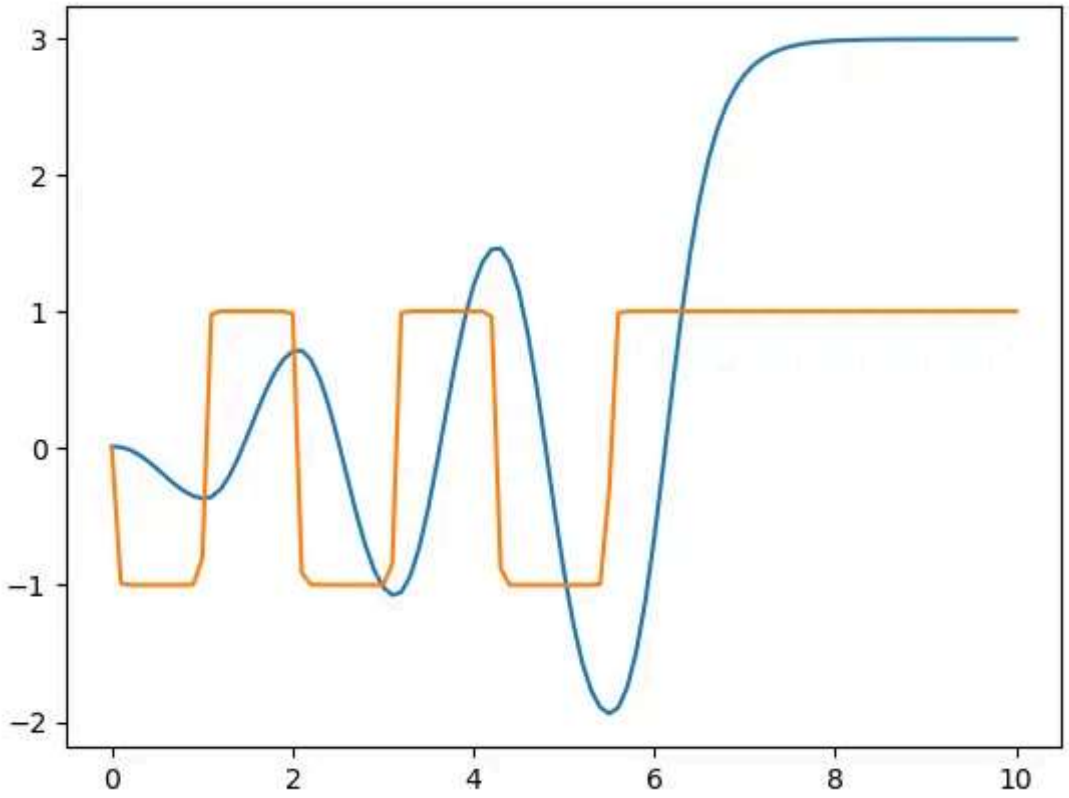


Fig b) Output join angles and torques for b=0.15

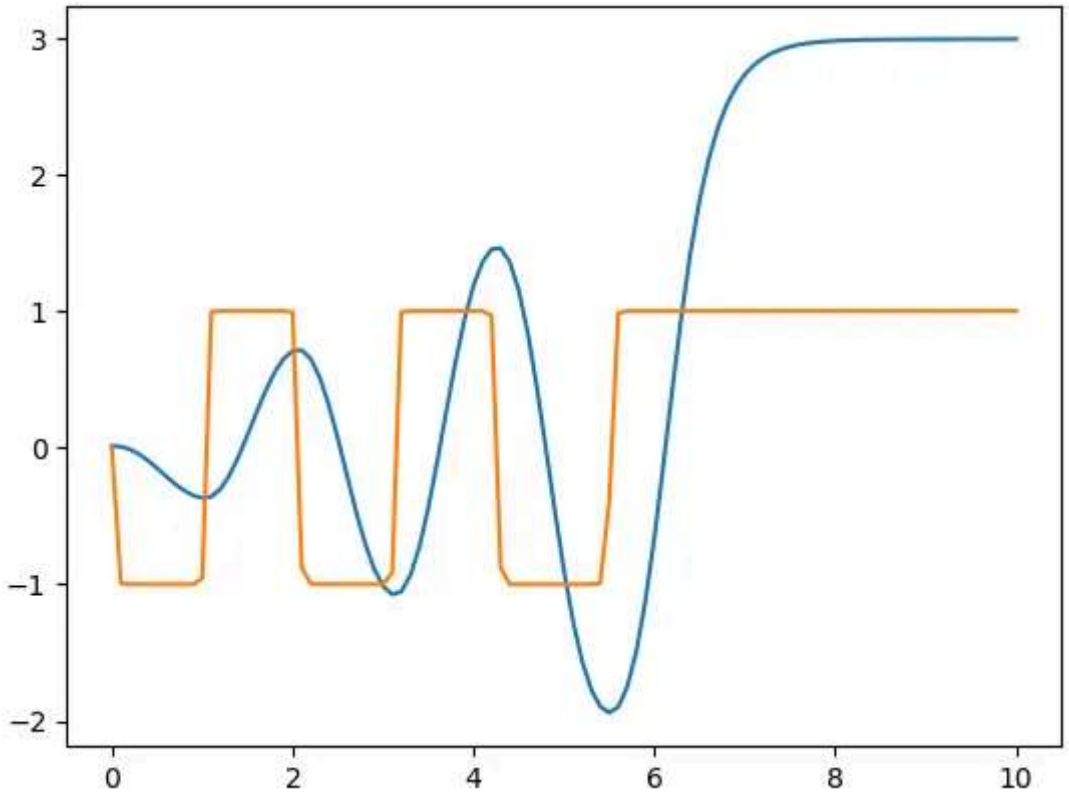


Fig c) Output torques and joint angles for b=0.15 and gaussian noise with σ = 0.01.

8.Conclusion

Physics-informed neural networks (PINNs) are a class of machine learning models that incorporate physical laws and perform tasks such as optimization and control

in physical systems. In this work, we applied PINNs to three distinct optimization tasks, pendulum swing-up, shortest-time path, and minimal-thrust swingby trajectories. The existing code was perfectly reproduced, and results and plots validated. Additionally, hyperparameter tuning and ablation studies were carried out deeming the existing hyperparameters and architecture optimal. Finally, the model robustness was validated for the inverting pendulum task by adding friction and gaussian noise.

Follow

Written by Manan Lohani

0 Followers

Recommended from Medium