ME228: Applied Data Science and Machine Learning

Discovering Governing Dynamical Systems using Physics Informed Neural Networks

Guide:
Prof. Shyamprasad Karagadde

Submitted By:

Aansh Samyani 22B0424

Susmit Neogi 22B0352

Abstract

- We have implemented Physics Informed Neural Networks for discovering underlying governing differential equations in physical phenomenon.
- We have implemented PINNs for approximating 4 physical phenomenon:
- 1. Lorenz System
- 2. Cylindrical Fluid Flow
- 3. Euler Rigid Body Dynamics
- 4. 2-Dimensional Pendulum Motion

We have taken inspiration from the SINDy (Sparse Identification of Nonlinear Dynamical Systems) Model which is used for discovering governing equations from data.

Physics Informed Neural Networks (PINNs)

Physics-Informed Neural Networks (PINNs) are a type of neural network architecture that incorporates physical principles or laws into their design to improve learning efficiency and accuracy, especially when working with limited or noisy data. PINNs are often used to discover or approximate governing equations from observational data or simulations.

How is this achieved?

PINNs are trained using a combination of data-driven techniques and physics-based constraints. This means that during training, the network also tries to satisfy the underlying physics equations that govern the system.
The loss function used in PINNs typically consists of two components:

Model Loss: This component measures the error between the predicted values and the actual data points. It is similar to the loss function used in standard supervised learning.

Physics Loss: This component enforces the network to satisfy the governing equations of the system. This loss can be from physics based constraints like Initial Conditions, Boundary Conditions, etc.

General Workflow

- For all the four subjects, we have followed the same general workflow:
- Define the system using state equations governing the physical phenomenon.
- Using these state equations, generate synthetic time dependent data.
- Add Gaussian Random Noise in the data to replicate real life scenario.
- Define a Neural Network architecture.
- Train the NN Model on the data generated.
- Compare the predicted system with actual system through 2D and 3D plots.

Lorenz System

- The Lorenz system is a set of three nonlinear ordinary differential equations (ODEs) that describe the behavior of a simplified model of atmospheric convection
- The Lorenz system is famous for exhibiting chaotic behavior, meaning that the system is highly sensitive to initial conditions.
- The Lorenz system dynamics is shown below:

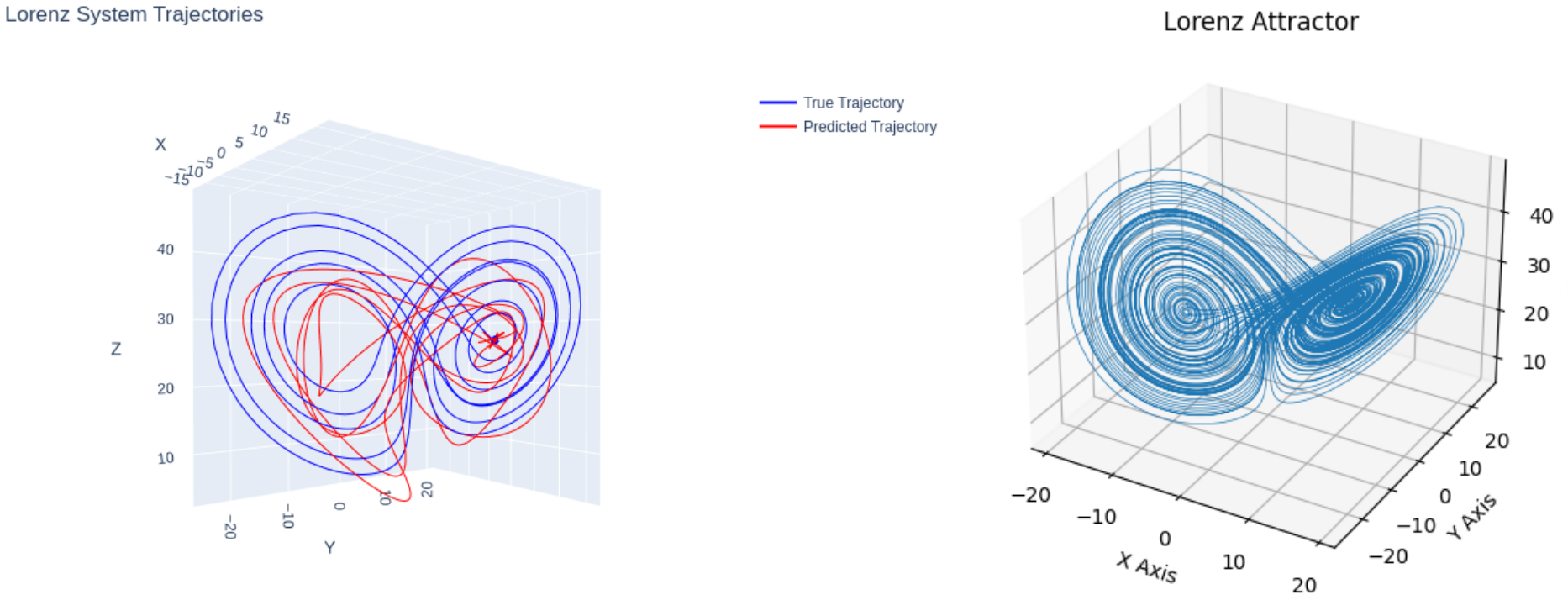
$$\begin{aligned} \dot{x} &= \sigma(y - x) \\ \dot{y} &= x(\rho - z) - y \\ \dot{z} &= xy - \beta z. \end{aligned}$$

For predicting the Lorenz System, we use the following techniques:

Regression Fit: We can approximate the Lorenz system as a Linear Ordinary Differential Equation in x, y,z and t. So using simple Linear regression fit, we can determine the coefficients of the differential equation.

Using PINN: We used a NN architecture with input layer of input dimension =1 and output dimension =20. Then added a hidden layer of input dimension =20 and output dimension =20. Finally, output layer of input dimension =20 and output dimension =3. Clearly, for this neural network, the input feature is time vector and output features are the predicted coordinates (x,y,z).
After each layer we added a Sine function as activation function. This is because our hypothesis is that most natural phenomenon follow an underlying sinusoidal relation.

For training the neural network, we used sum of Model Loss and Physics Loss. Model Loss is basically Mean Squared Error of Predicted state and actual state. Physics Loss is the physics constraint of Initial Conditions.
After training for 10000 epochs, we obtained a loss of 7.93.



Visualisation of Predicted and Actual System in 3D Space

Cylindrical Fluid Flow

- Modeling the wake behind a cylinder in a fluid flow, especially using mean field models, typically involves considering the flow as a combination of mean flow and turbulent fluctuations.
- While the full Navier-Stokes equations can describe this flow completely, mean field models aim to simplify the description by averaging out some of the complexities.

For predicting the fluid flow dynamics, we use a deeper Neural Network. This is because the mean field model is relatively more complex than the Lorenz system. The mean field model is shown below:

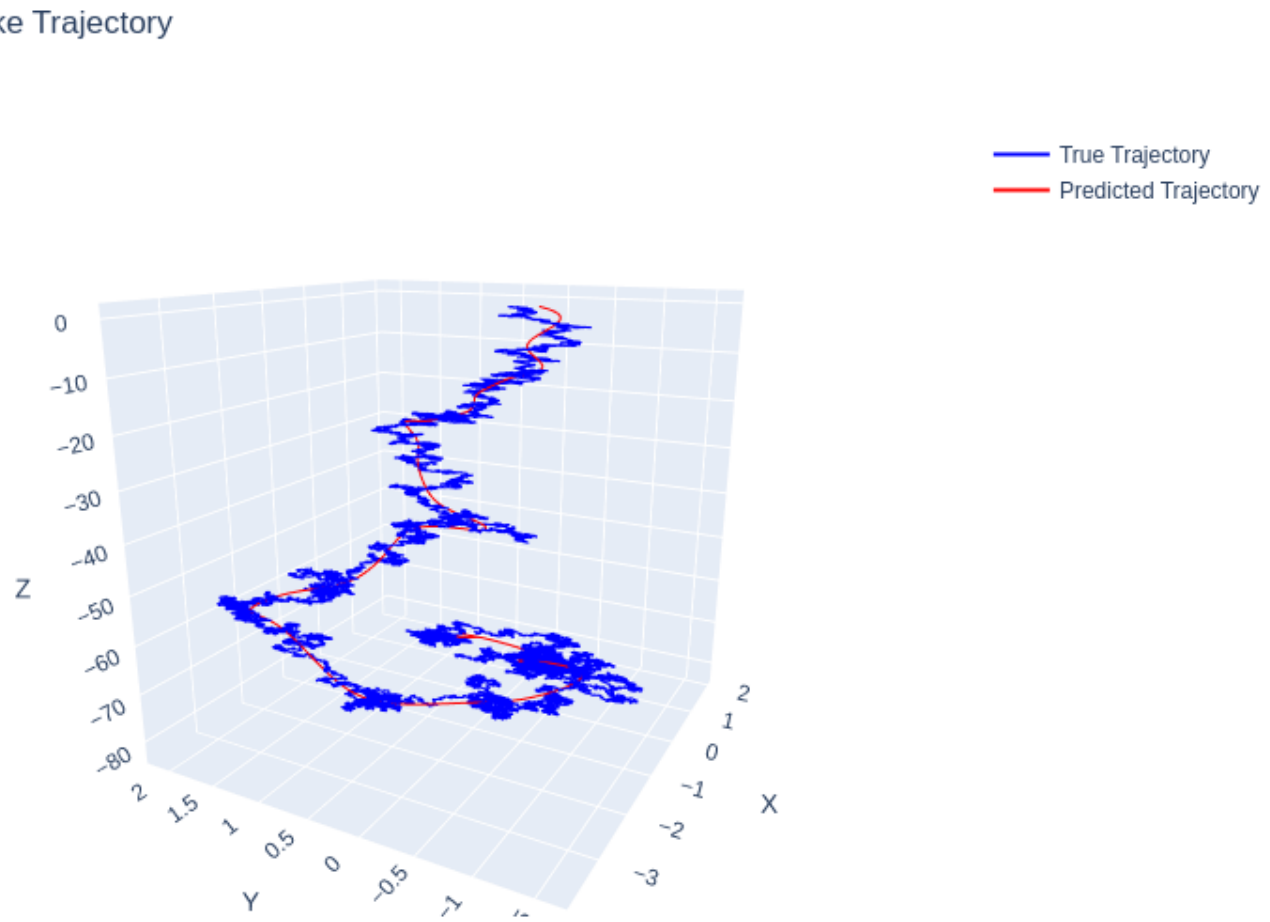
$$\frac{dx}{dt} = A_1x + B_1y + C_1z + D_1x^2 + E_1xy + F_1xz + G_1y^2 + H_1yz + I_1z^2$$

$$\frac{dy}{dt} = A_2x + B_2y + C_2z + D_2x^2 + E_2xy + F_2xz + G_2y^2 + H_2yz + I_2z^2$$

$$\frac{dz}{dt} = A_3x + B_3y + C_3z + D_3x^2 + E_3xy + F_3xz + G_3y^2 + H_3yz + I_3z^2$$

Our NN architecture has input layer of input dimension =1 and output dimension =20. Then a hidden layer of input dimension =20 and output dimension =120. Then another hidden layer of input dimension =120 and output =20. Finally, output layer of input dimension =20 and output dimension =3. Clearly, for this neural network, the input feature is time vector and output features are the predicted coordinates (x,y,z).

For training the neural network, we used sum of Model Loss and Physics Loss. Model Loss is basically Mean Squared Error of Predicted state and actual state. Physics Loss is the physics constraint of Initial Conditions.
After training for 50000 epochs, we obtained a loss of 0.0719.



Visualisation of Actual and predicted Cylindrical Fluid Wake Trajectory

Euler Rigid Body Dynamics

- Euler's equations describe the rotational motion of a rigid body, which is a body that maintains a constant shape and size, and where the relative position of all particles remains constant. Euler's equations are fundamental in classical mechanics and are derived from Newton's laws applied to rotational motion.
- These equations describe the rotational motion of a rigid body in terms of its moments of inertia, angular velocities, angular accelerations, and external torques. They are nonlinear differential equations and can be solved numerically for specific cases to determine the motion of the rigid body

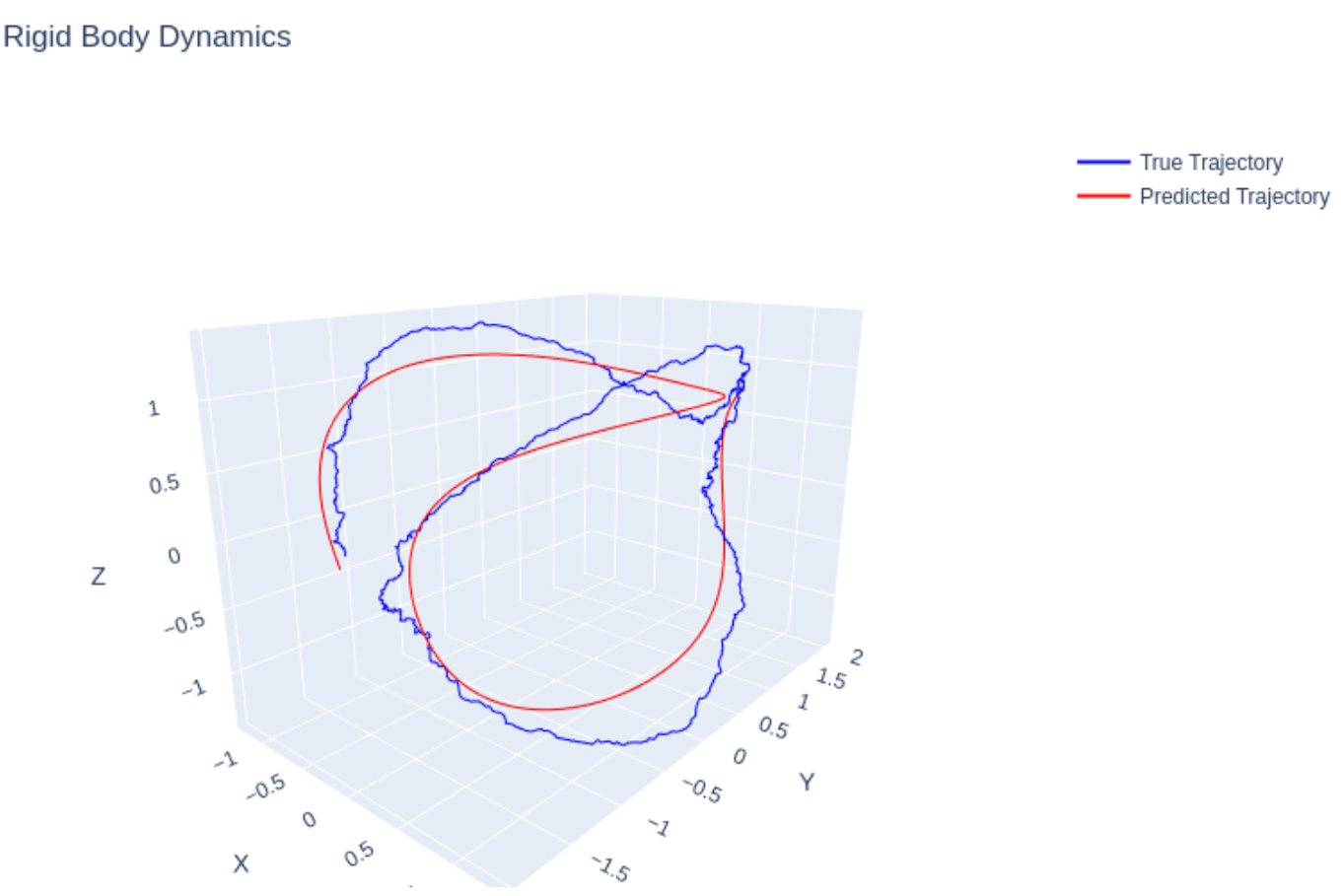
For predicting the Rigid Body Dynamics, we use the same Neural Network Architecture as for predicting Lorenz system because both of these dynamical systems are of almost same complexity. The Rigid Body Dynamics is given by the following equations:

$$\frac{dw_1}{dt} + (I_3 - I_2)w_2w_3 = 0$$

$$\frac{dw_2}{dt} + (I_1 - I_3)w_3w_1 = 0$$

$$\frac{dw_3}{dt} + (I_2 - I_1)w_2w_2 = 0$$

We used a NN architecture with input layer of input dimension =1 and output dimension =20. Then added a hidden layer of input dimension =20 and output dimension =20. Finally, output layer of input dimension =20 and output dimension =3. Clearly, for this neural network, the input feature is time vector and output features are the angular velocities (w1, w1, w3).
For training the neural network, we used sum of Model Loss and Physics Loss. Model Loss is basically Mean Squared Error of Predicted state and actual state. Physics Loss is the physics constraint of Initial Conditions.
After training for 50000 epochs, we obtained a loss of 0.077.



Visualisation of actual trajectories and predicted trajectories

2-D Oscillating Pendulum

- A 2D pendulum refers to a simple pendulum that moves in two dimensions, typically in a vertical plane.
- This system describes the motion of the 2D pendulum influenced by gravity and the lengths of the pendulum arms. The equations are coupled because the motion in one direction affects the motion in the other direction.
- The equations of the system are shown below:

$$\frac{d^2\theta}{dt^2} + \frac{g}{\ell}\sin\theta = 0$$

In this problem, for predicting the dynamical system, we have used a different approach. Please note that we have done only the initial analysis and architecture design for this problem.

We obtained the simulation images of a swinging 2D pendulum at different time stamps. Now we designed a Autoencoder, which takes these images (2601 pixels) as inputs and reduces the dimensions returns a single feature. This feature probably represents the angle moved θ with the vertical axis.

Now we train our decoder model. It takes in the extracted feature from the encoder and returns the original state, that is, the pendulum position image. For training this decoder, we use 4 Loss functions:

1. **Reconstruction Loss**: Loss between decoder output image and original image.
2. **SINDy Loss in X**: Loss due to predicted and actual equations in X.
3. **SINDy Loss in Z**: Loss due to predicted and actual equations in Z.
4. **Regularisation Loss**: This loss penalizes for having too many non-zero coefficients,thus encouraging a simpler and more interpretable model