Effect of Different Machine Learning Algorithms on Predicting the Trajectory of a Charged Particle in an Electromagnetic Field

Ahmad_Aliahmad

May 2025

Contents

1 Problem Statement

Given the trajectory data of a charged particle in an electromagnetic field, we would like to predict its Future states using SympNets [2], PNNs [1], and PINNs [3]. The motion of the particle is governed by The Lorentz force:

$$m\ddot{x} = q(E + \dot{x} \times B)$$

where m is the mass, $x \in \mathbb{R}^3$ denotes the particle's position, q is the electric charge, $B = \nabla \times A$ denotes the magnetic field, and $E = -\nabla\varphi$is the electric field with A, φ being the potentials. Let $\dot{x} = v$ be the velocity of he charged particle, then the governing equations of the particle's motion can be expressed as

$$\begin{pmatrix} \dot{v} \\ \dot{x} \end{pmatrix} = \begin{pmatrix} -\frac{q}{m^2}\hat{B} & -\frac{1}{m}I \\ \frac{1}{m}I & 0 \end{pmatrix} \nabla H(v, x),$$

$$H(v, x) = \frac{1}{2}mv^T v + q\varphi(x),$$

where

$$\hat{B}(x) = \begin{pmatrix} 0 & -B_3(x) & B_2(x) \\ B_3(x) & 0 & -B_1(x) \\ -B_2(x) & B_1(x) & 0 \end{pmatrix}$$

for $B(x) = (B_1(x), B_2(x), B_3(x))$. Here we test the dynamics with $m = 1, q = 1$, and

$$A(x) = \frac{1}{3}\sqrt{x_1^2 + x_2^2} \cdot (-x_2, x_1, 0), \varphi(x) = \frac{1}{100\sqrt{x_1^2 + x_2^2}}$$

for $(x_1, x_2, x_3)^T$. Then

$$B(x) = (\nabla \times A)(x) = (0, 0, \sqrt{x_1^2 + x_2^2})$$

$$E(x) = -(\nabla\varphi)(x) = \frac{(x_1, x_2, 0)}{100(x_1^2 + x_2^2)^{\frac{3}{2}}}$$

2 Abstract

The main goal of this experiment is to use a dataset of 1500 data points (1200 for training, 300 for testing) generated by a Stormer-Verlet integrator, with each data point having a step size of $h = 0.1$. We need to find a function approximator using three different neural network structures (SympNets, PNNs, and PINNs). We train the networks to predict the trajectory of the particle from a single initial state.

3 Methodology

3.1 SympNet

3.1.1 Theory

Symplectic neural networks are designed to take advantage of a system's symplecticity to create an accurate map. So a SympNet would be able to solve constant Hamiltonian systems in the form,

$$\dot{y} = J^{-1}\nabla H(y)$$

The main idea of this architecture is to learn the symplectic design of the Hamiltonian system. This is possible only because of the characteristics of the constant hamiltonian, which tell us that there exists a symplectic map that can fully model the system. So if we can ensure that the neural network can keep the symplectic properties of the input and output, it would make learning the system from the inputs and outputs of the trajectory possible. This is done by layering different symplectic matrices on top of each other to enforce the symplecticity of the network. There are 2 types of Sympnets depending on what layers you choose to keep in your neural network.

3.1.2 Loss

The loss function will be a general Mean Squared Error function.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

N being the batch size $\hat{y}$ is the predicted and y is the true value. We get the average and backpropagate based on that value.

3.1.3 LA-SympNet

This symplectic map adopts 2 of the 3 main modules. The linear modules defined by

$$L_n \begin{pmatrix} p \\ q \end{pmatrix} = \begin{pmatrix} I & 0/S_n \\ S_n/0 & I \end{pmatrix} ... \begin{pmatrix} I & 0 \\ S_2 & I \end{pmatrix} \begin{pmatrix} I & S_1 & \\ 0 & & I \end{pmatrix} \begin{pmatrix} p \\ q \end{pmatrix} + b$$

$$, q \in \mathbb{R}^d$$

Where $S_i \in \mathbb{R}^{b\times b}$ are symmetric,$b \in \mathbb{R}^{2d}$is the bias, while the unit upper triangular symplectic matrices and the unit lower triangular symplectic matrices appear alternately. In this module,S_i (represented by $A_i + A_i^T$ in practice) and b are parameters to learn. In fact L_n can represent any linear symplectic map.
The activation modules

$$N_{up} = \begin{pmatrix} p \\ q \end{pmatrix} = \begin{pmatrix} p + \hat{\sigma}_a(q) \\ q \end{pmatrix} \quad N_{low} \begin{pmatrix} p \\ q \end{pmatrix} = \begin{pmatrix} p \\ \hat{\sigma}_a(p) + q \end{pmatrix}, \quad p, q \in \mathbb{R}^d$$

where $\hat{\sigma}_a(x)a \odot \sigma(x)$ for $x \in \mathbb{R}^d$. Here $\odot$ is the element-wise product, σ is the activation function, and $a \in \mathbb{R}^d$ is the parameter to learn

These 2 modules represent the architecture of the neural network. A combination of the up and down types of either module will represent the entire constant Hamiltonian.

3.1.4 G-SymptNet

The gradient module

$$G_{up} \begin{pmatrix} p \\ q \end{pmatrix} = \begin{pmatrix} p + \hat{\sigma}_{K,a,b}(q) \\ q \end{pmatrix}$$

$$G_{low} \begin{pmatrix} p \\ q \end{pmatrix} = \begin{pmatrix} p \\ \hat{\sigma}_{K,a,b}(p) + p \end{pmatrix}, \ p, q \in \mathbb{R}^d$$

where $\hat{\sigma}_{K,a,b}(x) := K^T(a \odot \sigma(Kx + b))$ for $x \in \mathbb{R}^d$. Here $a, b \in \mathbb{R}^l, K \in \mathbb{R}^{l\times d}$ are the parameters to learn and l is a positive integer regarded as the width of the module.

3.2 PNN

3.2.1 Theory

PNN and SympNets are very similar to one another, but allow us to solve non-constant Hamiltonian systems; in other words they have looser constraints than the SympNets, which includes the problem we are currently tackling. The way we solve this is by transforming the coordinates of our problem, which isn't symplectic, into becoming symplectic. We do this using the local Euclidean property of manifolds. A topological manifold is a space that locally resembles Euclidean space ($\mathbb{R}^n$). This means that for every point on the manifold, there's a neighborhood that can be mapped (via a homeomorphism) to $\mathbb{R}^n$. This allows us to zoom into our problem so much that we can assume that the space we are working on is symplectic. Then, because we can use an invertible transformer to get the inverse of that, we get back to our known coordinates.

3.2.2 Architecture

The basic idea of this architecture would look like this.

$$L(\tau) = \frac{1}{n \cdot N} \sum_{i=1}^{N} ||\theta^{-1} \circ \Phi \circ \theta(x_i) - y_i||^2$$

where $\phi : \mathbb{R}^n \to \mathbb{R}^n$ is an extended symplectic neural network with latent dimension $2d$ and θ is a volume preserving transformer which means that

$$\theta^{-1}(\theta(x)) = x$$

Which would keep its injective property, which is important for symplecticity.

3.3 NICE Coupling layer

NICE is a basic volume-preserving invertible neural network that is made up of these modules

$$V_{up} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1 + m_1(x_2) \\ m_2(x_1) + x_2 \end{pmatrix}$$

$$V_{low} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1 \\ m_2(x_1) + x_2 \end{pmatrix}$$

$$x_1 \in \mathbb{R}^d, x_2 \in \mathbb{R}^{n-d}$$

where $m_1 : \mathbb{R}^{\ltimes -} \to \mathbb{R}^d$ and $m_2 : \mathbb{R}^d \to \mathbb{R}^{n-d}$ are modeled as fully connected neural networks. These modules will be acting as our θ and θ^{-1} that we mentioned in our primary architecture above.

3.4 Extended SympNet

The extended Sympnet is just a further developed G-SympNet, as you would have guessed based on the definition.

$$\mathcal{E}_{up} \begin{pmatrix} p \\ q \\ c \end{pmatrix} = \begin{pmatrix} p + \hat{\sigma}_{K_1,K_2,a,b}(q,c) \\ q \\ c \end{pmatrix} \quad \mathcal{E}_{low} \begin{pmatrix} p \\ q \\ c \end{pmatrix} = \begin{pmatrix} p \\ \hat{\sigma}_{K_1,K_2,a,b}(q,c) + q \\ c \end{pmatrix}$$

where $\hat{\sigma}_{K_1,K_2,a,b}(x,c) := K_1^T(a \odot \sigma(K_1 x + K_2 c + b))$ for $x \in \mathbb{R}^d, c \in \mathbb{R}^{n-2d}$. Here $a, b \in \mathbb{R}^l, K_1 \in \mathbb{R}^{l \times d}, K_2 \in \mathbb{R}^{l \times (n-2d)}$ are the parameters to learn, and l is a positive integer regarded as the width of the module.
The E-SympNet is just a composition of these extended modules just just like how we saw in the other Sympnets.

3.5 Loss

The loss function will be a general Mean Squared Error function.

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y_i})^2$$

N being the batch size $\hat{y}$ is the predicted and y is the true value. We get the average and backpropagate based on that value.

3.6 PINN

3.6.1 Theory

Physics-informed neural networks are mainly used to solve a PDE and find the variables that would construct a solution to the equation itself. In this case, it would be the q and m of the particle. Using a neural network to learn the Hamiltonian and using the rest of the PDE equation to find the next state using automatic differentiation and the RK4 integrator.

3.6.2 Architecture

The architecture of this neural network comes from an equation we discussed at the start of this paper.

$$\begin{pmatrix} \dot{v} \\ \dot{x} \end{pmatrix} = \begin{pmatrix} -\frac{q}{m^2}\hat{B} & -\frac{1}{m}I \\ \frac{1}{m}I & 0 \end{pmatrix} \nabla H(v, x),$$

A neural network will represent the Hamiltonian then we get the gradient using automatic differentiation. The dynamic matrix can be constructed with the lambdas representing ratios between m and q. The major difference in this neural network is the loss calculations, one being the actual loss, which trains the Hamiltonian, and the residual loss, which finds the lambdas of the PDE.

3.6.3 Residual Loss

The residual loss is the physics loss of the network, where we use the known PDE equation to calculate it. This is, of course, the governing equation that we discussed earlier. The residual loss for this equation can be summed up here.

$$\begin{pmatrix} \dot{v} \\ \dot{x} \end{pmatrix} = \begin{pmatrix} 0 & \lambda_1\lambda_2\sqrt{x_1^2 + x_2^2} & -\lambda_2 & 0 \\ -\lambda_1\lambda_2\sqrt{x_1^2 + x_2^2} & 0 & 0 & -\lambda_2 \\ \lambda_2 & 0 & 0 & 0 \\ 0 & \lambda 2 & 0 & 0 \end{pmatrix} \nabla \mathcal{L}(v, x)$$

$$Loss_R = \frac{1}{N}\sum_{i=0}^{N} \left(\int \begin{pmatrix} \dot{v} \\ \dot{x} \end{pmatrix}_i - \hat{y}_i \right)^2$$

$\mathcal{L}$ representing the neural network, and we get its gradient using automatic differentiation. $\lambda_1 = \frac{q}{m}$ and $\lambda_2 = \frac{1}{m}$. Then, using the RK4 integrator on the output tensor, we get the next state. This is only used for residual loss calculations or after the model has been trained.

3.6.4 Actual Loss

The actual loss of the neural network can be calculated using the training data and is like the other losses that we discussed in this paper. It is used to ground the Hamiltonian training and accelerate the convergence of the lambdas.

$$Loss_A = \frac{1}{N}\sum_{i=0}^{N}(y_i - \hat{y}_i)^2$$

4 Result

In this section, we will discuss the results of the traditional Architectures. The tests were conducted the same way. The neural network was given an initial state, and from there it was supposed to predict the next 299 steps without any extra input. The testing data was confirmed using our 300 testing data points that were mentioned before. Each model was trained approximately 50,000epochs with a batch size of 128. The learning rate was on a scheduler that would drop it if a plateau was sensed.
For the exact numbers of layers, activation functions, etc... The implementation can be found on the GitHub page found in the Section.

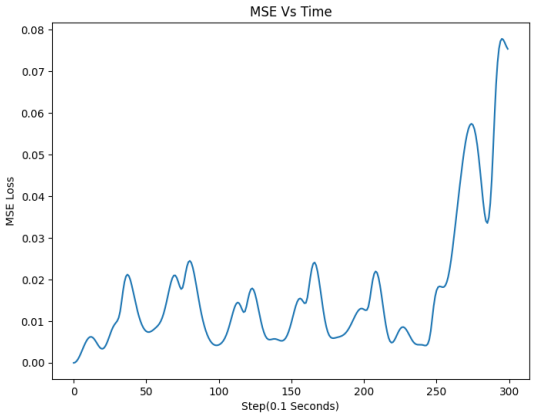

Figure 1: MSE loss during training for LA-SympNet

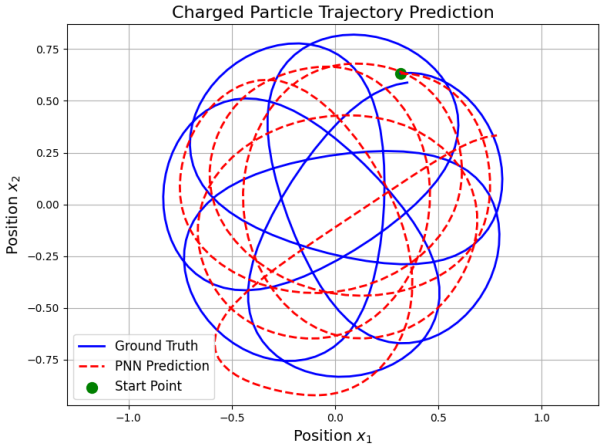

Figure 2: Trajectory prediction using LA-SympNet

4.1 SympNet

4.1.1 LA-SympNet

As shown in Figures 1 and 2, LA-SympNet failed to converge despite extensive hyperparameter tuning. This aligns with expectations because SympNets are fundamentally designed for constant Hamiltonian systems, while our problem involves a position-dependent magnetic field.

4.1.2 G-SympNet

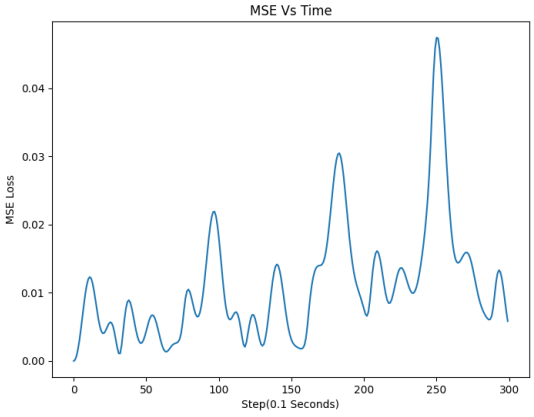

Figure 3: MSE loss during training for G-SympNet

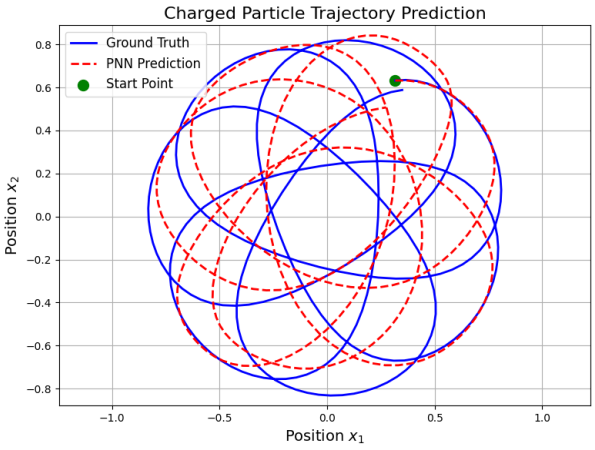

Figure 4: Trajectory prediction using G-SympNet

Figures 3 and 4 demonstrate improved performance compared to LA-SympNet. The gradient modules provide flexibility for handling non-constant Hamiltonians, though residual errors persist due to fundamental architectural constraints.

4.2 PNN

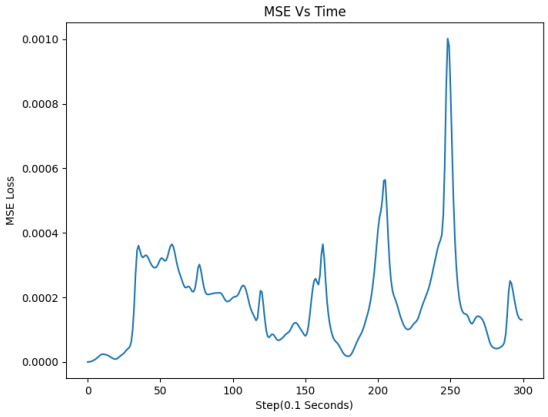

Figure 5: MSE loss during training for PNN

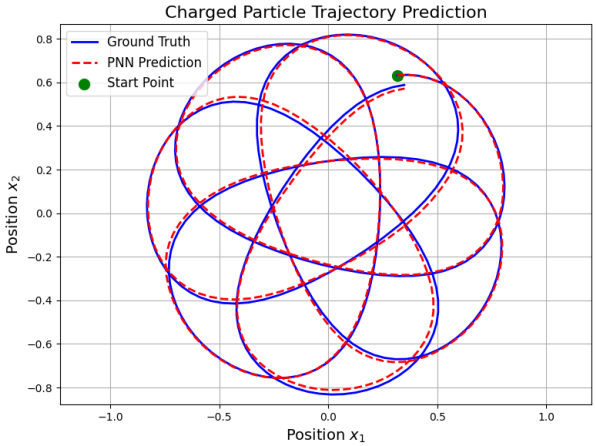

Figure 6: Trajectory prediction using PNN

The PNN architecture (Figures 5 and 6) shows dramatic improvement, successfully accounting for the position-dependent $\hat{B}(x)$ matrix through coordinate transformation. This validates our hypothesis about manifold embedding for non-constant Hamiltonians.

4.3 PINN

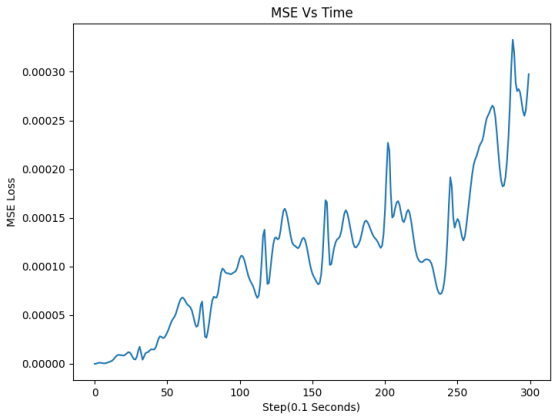

Figure 7: MSE loss during training for PINN

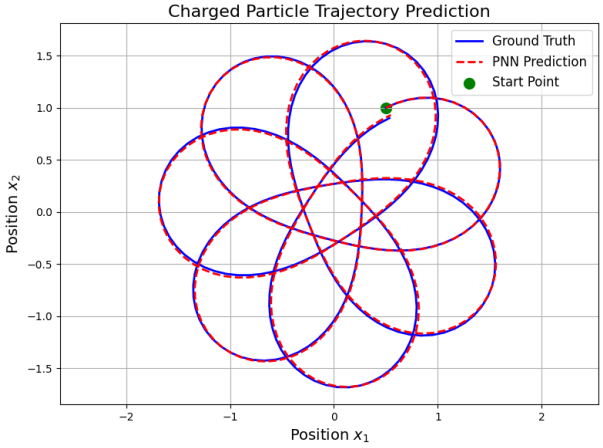

Figure 8: Trajectory prediction using PINN

As seen in Figures 7 and 8, the PINN achieves near-perfect convergence, correctly identifying the $\frac{q}{m} = 1$ ratio through combined data-driven and physics-constrained learning. The residual loss formulation effectively regularizes the Hamiltonian approximation.

5 Conclusion

In this study, we evaluated three neural network architectures for predicting charged particle trajectories in electromagnetic fields:

- **SympNets** demonstrated fundamental limitations for non-constant Hamiltonian systems, as expected from their theoretical framework. While LA-SympNet failed to converge (Figure 2), G-SympNet showed modest improvement (Figure 4), confirming the need for more flexible architectures when dealing with position-dependent fields.
- **PNNs** emerged as a robust alternative, successfully handling the non-constant Hamiltonian through coordinate transformations. The architecture's ability to recover from prediction errors (Figure 6) suggests particular value in scenarios where complete physical constraints are unavailable.
- **PINNs** achieved the highest accuracy when physical constraints were incorporated through residual loss (Figure 8). However, their sensitivity to initial conditions, evident in the error accumulation shown in Figure 7, suggests caution in long-term predictions.

These results lead to three key recommendations for practitioners:

1. When governing equations are known and can be incorporated as constraints, PINNs provide optimal accuracy
2. For systems with unknown or complex constraints, PNNs offer a balance of flexibility and performance
3. SympNets should be reserved for systems with truly constant Hamiltonians

Future work could explore hybrid architectures combining the strengths of these approaches, particularly for multi-scale physical systems where different regimes may benefit from different modeling paradigms.

References

[1] P. Jin, Z. Zhang, I. G. Kevrekidis, and G. E. Karniadakis. *Learning Poisson systems and trajectories of autonomous systems via Poisson neural networks*. arXiv preprint arXiv:2012.03133, 2020.

[2] P. Jin, Z. Zhang, A. Zhu, Y. Tang, and G. E. Karniadakis. *SympNets: Intrinsic structure-preserving symplectic networks for identifying Hamiltonian systems*. Neural Networks, 132:166–179, 2020.

[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, 378:686–707, 2019.

[4] Zhang, Z. *Course project: Charged particle in an electromagnetic field*. Dropbox.com. Retrieved May 11, 2025, from `https://www.dropbox.com/scl/fo/87ks8c9jd6ws1eyaar61k/AOhTyfXvLNtRNNviiX1gpWI?e=6&preview=Course_project__Charged_particle_in_a_electromagnetic_field.pdf&rlkey=g3gal8ufvo9fodu6z1f76kngt&dl=0`

6 Appendix

All the code mentioned above can be found here. It was coded in Pytorch.
`https://github.com/AhmadAli2024/Charged-particle-in-a-electromagnetic-field`