# SYMPLECTIC NEURAL NETWORKS IN TAYLOR SERIES FORM FOR HAMILTONIAN SYSTEMS

**Yunjin Tong**[a*], **Shiying Xiong**[a†], **Xingzhe He**[a,b], **Guanghan Pan**[a,c], **Bo Zhu**[a]
$a$ Dartmouth College, Hanover, NH 03755, United States
$b$ Rutgers University, New Brunswick, NJ 08854, United States
$c$ Middlebury College, Middlebury, VT 05753, United States

April 22, 2021

**ABSTRACT**

We propose an effective and light-weight learning algorithm, Symplectic Taylor Neural Networks (Taylor-nets), to conduct continuous, long-term predictions of a complex Hamiltonian dynamic system based on sparse, short-term observations. At the heart of our algorithm is a novel neural network architecture consisting of two sub-networks. Both are embedded with terms in the form of Taylor series expansion designed with symmetric structure. The key mechanism underpinning our infrastructure is the strong expressiveness and special symmetric property of the Taylor series expansion, which naturally accommodate the numerical fitting process of the gradients of the Hamiltonian with respect to the generalized coordinates as well as preserve its symplectic structure. We further incorporate a fourth-order symplectic integrator in conjunction with neural ODEs' framework into our Taylor-net architecture to learn the continuous-time evolution of the target systems while simultaneously preserving their symplectic structures. We demonstrated the efficacy of our Taylor-net in predicting a broad spectrum of Hamiltonian dynamic systems, including the pendulum, the Lotka–Volterra, the Kepler, and the Hénon–Heiles systems. Our model exhibits unique computational merits by outperforming previous methods to a great extent regarding the prediction accuracy, the convergence rate, and the robustness despite using extremely small training data with a short training period (6000 times shorter than the predicting period), small sample sizes, and no intermediate data to train the networks.

**Keywords**: Machine learning, Hamiltonian system, Physics-informed neural network, Taylor series expansion

## 1 Introduction

Hamiltonian mechanics, first formulated by William Rowan Hamilton in 1834 [18], is one of the most fundamental mathematical tools for analyzing the long-term behavior of complex physical systems studied over the past centuries [46, 11]. Hamiltonian systems are ubiquitous in nature, exhibiting total energy with various forms, as seen in plasma physics [30], electromagnetic physics [27], fluid mechanics [39], and celestial mechanics [38]. Mathematically, Hamiltonian dynamics describe a physical system by a set of canonical coordinates, i.e., generalized positions and generalized momentum, and uses the conserved form of the symplectic gradient to drive the temporal evolution of these canonical coordinates [19]. However, for a dynamic system governed by some unknown mechanics, it is challenging to identify the Hamiltonian quantity and its corresponding symplectic gradients by directly observing the system's state, especially when such observation is partial and the sample data is sparse [15, 3, 43].

The rapid advent of machine learning (ML) techniques opens up new possibilities to solve the identification problems of physical systems by statistically exploring their underlying structures. On the one hand, data-driven approaches

*Co-first author
†Co-first author, shiying.xiong@dartmouth.edu

have proven their efficacy in uncovering the underlying governing equations of a variety of physical systems, encompassing applications in fluid mechanics [2], wave physics [23], quantum physics [41], thermodynamics [21], and material science [44]. On the other hand, various ML methods have been proposed to boost the numerical simulation of complex dynamical systems by incorporating learning paradigms into simulation infrastructures, e.g., ordinary differential equations [35], linear or nonlinear partial differential equations [47, 35, 32, 31, 22, 36], high-dimensional partial differential equations [42], inverse problems [34], space-fractional differential equations [16], systems with noisy multi-fidelity data [33], and pseudo-differential operators [10, 9], to name a few. More recently, many lines of research have tried to incorporate physical priors into the learning framework, instead of letting the learning algorithm start from scratch, e.g., embedding the notion of an incompressible fluid [29, 48], the Galilean invariance [28], a quasistatic physics simulation [13], and the invariant quantities in Lagrangian systems [7] and Hamiltonian systems [21, 14, 24, 51, 8, 49].

There are two critical aspects in learning and predicting the dynamics of a Hamiltonian system. The first key point is to learn the continuous dynamic time evolution. It is impossible to control the growth of approximation error and monitor the level of error by simply using neural networks to learn the dynamics of a system and integrating using traditional integrators, e.g., Euler [17], Runge–Kutta [37, 26]. Secondly and also more challengingly, finding the symplectic gradients that have symmetric structure is hard. The exact solution of a Hamiltonian system leads to a symplectic map from the initial conditions to an arbitrary present state. Due to inaccuracies arising from the computed gradients of a high-dimensional Hamiltonian using traditional neural networks, finding the exact structure of the symplectic gradients from non-differentiable functions will often cause a large error. To address these two critical aspects, we propose the following solutions. Firstly, we utilize the neural ODE (ODE-net)'s framework, introduced by Chen et al. in 2018, [5], to obtain the continuous evolution. Drawing parallels between residual neural networks [20] and the modeling pattern of an ODE, Chen et al. utilize continuously-defined dynamics to naturally incorporate data that arrive at arbitrary times. The main difficulty lies in addressing the second aspect. To preserve symplectic structure while accurately approximating the continuous-time evolution of dynamical systems, the neural networks have to fulfill two criteria:

1. The gradients of the Hamiltonian with respect to the generalized coordinates should be symmetric.
2. The temporal integration should be symplectic.

We made two essential contributions to meet the above two criteria when processing a Hamiltonian system by incorporating a set of special computing primitives into traditional neural networks. First, to enable symmetric gradients of the Hamiltonian with respect to the generalized coordinates, we construct neural networks that model the gradients and preserve their symmetric structure. Due to the multi-nonlinear-layer architecture of traditional deep neural networks, it is impossible for these networks to fulfill the symmetric property. Thus, we can only use a three-layer network with the form of *linear-activation-linear*, where the weights of the two linear layers are the transpose of each other. However, such a shallow network cannot capture the complexity of Hamiltonian systems. Therefore, in order to maintain the expressive power of the network, we create multiple such three-layer sub-networks and combine them linearly into the Taylor series form. As a result, our network architecture naturally preserves the symmetry of the structure while exhibiting strong expressive power. Furthermore, to enable a symplectic preserving temporal evolution, we implement a fourth-order symplectic integrator [12, 52] within a neural ODE-net architecture [5, 53]. This fourth-order integration step enables an explicit fourth-order symplectic mapping to preserve the canonical character of the equations of motion in an exact manner. In other words, it preserves the property that the temporal evolution of a Hamiltonian system yields a canonical transformation from the initial conditions to the final state [12].

Based on these two major enhancements, we propose a novel neural network model, *symplectic Taylor neural networks (Taylor-nets)*, to precisely preserve the quantity and predict the dynamics of a Hamiltonian system. The Taylor-nets consist of two sub-networks whose outputs are combined using a fourth-order symplectic integrator. Both sub-networks are embedded with the form of Taylor series expansion and learn gradients of the position and momentum of the Hamiltonian system, respectively. We design the sub-networks such that each term of the Taylor series expansion is symmetric. The symmetric property of the terms and the fourth-order symplectic integrator ensure our model intrinsically preserves the symplectic structure of the underlying system. Therefore, the prediction made by our neural networks leads to a symplectic map from an initial condition to the present state of a Hamiltonian system, which is the most fundamental feature of the exact solution of a Hamiltonian system.

With the integrated design of the sub-networks symmetric structure and the fourth-order symplectic integrator, our learning algorithm is capable of utilizing extremely limited training data to generate highly accurate predictive results that satisfy the conservation laws in various forms. In particular, we demonstrate that the training period of our model can be around 6000 times shorter than its predicting period (other methods have the training period 1–25 times shorter than the predicting period [5, 14, 24]), and the number of training samples is around 5 times smaller (meaning we use 5 times fewer time-sequences as in the training process) than that used by other methods. Moreover, our method only

requires the data collected at the two endpoints of the training period to train the neural networks, without requiring any intermediate data samples in between the initial point and the endpoint. These improvements are crucial for modeling a realistic, complex physical system because they minimize the requirement of training data, which are typically difficult to obtain, and reduce training time by a significant amount. Other major computational merits of our proposed method include its fast convergence rate and robustness. Thanks to the intrinsic structure-preserving characteristic of our method, our model converges more than 10 times faster than the other methods and is more robust under large noise. Overall, the contributions of our work can be summarized as below:

- We design a neural network architecture that intrinsically preserves the symplectic structure of the underlying system and predicts the continuous-time evolution of a Hamiltonian system.
- We embed the form of Taylor series expansion into the neural networks with each term of the Taylor series expansion designed to be symmetric.
- Our model outperforms other state-of-the-art methods regarding the prediction accuracy, the convergence rate, and the robustness despite using small data with a short training period, small sample sizes, and no intermediate data to train the model.

Our work is inspired by previous methodologies that incorporate the symplectic structure of a Hamiltonian system into neural networks. Greydanus et al. first tried to enforce conservative features of the Hamiltonian system by reformulating the loss function using Hamilton's equations, known as Hamiltonian neural networks (HNNs) [14]. Based on HNNs, many works were developed. Chen et al. developed symplectic recurrent neural networks (SRNN), which is a recurrent HNN that relies on a symplectic integrator [6]. Toth et al. developed the Hamiltonian Generative Network (HGN), learning Hamiltonian dynamics from high-dimensional observations (such as images) without restrictive domain assumptions [45]. Zhong introduced Symplectic ODE-Net (SymODEN), which adds an external control term to the standard Hamiltonian dynamics in order to learn the system dynamics which conform to Hamiltonian dynamics with control [51]. Methods like HNN, which focuses on the reformulation of the loss function, incur two main limitations. On the one hand, it requires the temporal derivatives of the momentum and the position of the systems to calculate the loss function, which is difficult to obtain from real-world systems. On the other hand, HNN doesn't strictly preserve the symplectic structure, because its symplectomorphism is realized by its loss function rather than its intrinsic network architecture. Our model successfully bypasses the time derivatives of the datasets by incorporating an integrator solver into the network architecture. Moreover, we design our model differently by embedding a symmetric structure into the neural networks, instead of manipulating the loss function. Thus, our model can strictly preserve the symplectic structure.

Independently, an intrinsic way to encode the symplectic structure is introduced by Jin et al. [24]. Such neural networks are called Symplectic networks (SympNets), which intrinsically preserve the symplectic structure for identifying Hamiltonian systems. Motivated by SympNets, we invent a neural network architecture to intrinsically preserve the symplectic structure. However, our model preserves two major advantages over SympNets. First, our model is capable of learning the continuous-time evolution of dynamical systems. Second, our model can easily be extended to N-body systems. The parameters scale in the matrix map for training $N$ dimensional Hamiltonian system of our model is $O(1)$. The number of parameters does not increase since based on the interactive models between particle pairs we only need data collected from two bodies as the training data to predict the dynamics of many bodies. However, SympNets require $O(N^2)$ complexity, which makes it hard to generalize to the high-dimensional N-body problems.

The structure of this paper is as follows. In section 2, we will first introduce the mathematical formulas and their proofs that serve as the foundation of our methodology. Then, we will discuss the design of our neural networks in Taylor series form as well as the proofs of their symplectic structure-preserving property. The next section 3 describes the implementation details and numerical results, which compare our methodology with other state-of-the-art methods, such as ODE-net and HNN. In section 4, we extend the application of our methodology to solve an N-body problem. Lastly, conclusions are drawn in a section 5 with discussions of potential directions of our future research.

## 2 Mathematical foundation

### 2.1 Hamiltonian mechanics

We start by considering a Hamiltonian system with $N$ pairs of canonical coordinates (i.e. $N$ generalized positions and $N$ generalized momentum). The time evolution of canonical coordinates is governed by the symplectic gradient of the Hamiltonian [19]. Specifically, the time evolution of the system is governed by Hamilton's equations as

$$\begin{cases} \dfrac{\mathrm{d}\boldsymbol{q}}{\mathrm{d}t} = \dfrac{\partial \mathcal{H}}{\partial \boldsymbol{p}}, \\ \dfrac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}t} = -\dfrac{\partial \mathcal{H}}{\partial \boldsymbol{q}}, \end{cases} \tag{1}$$

with the initial condition

$$(\boldsymbol{q}(t_0), \boldsymbol{p}(t_0)) = (\boldsymbol{q}_0, \boldsymbol{p}_0). \tag{2}$$

In a general setting, $\boldsymbol{q} = (q_1, q_2, \cdots, q_N)$ represents the positions and $\boldsymbol{p} = (p_1, p_2, ...p_N)$ denotes their momentum. Function $\mathcal{H} = \mathcal{H}(\boldsymbol{q}, \boldsymbol{p})$ is the Hamiltonian, which corresponds to the total energy of the system. By assuming that the Hamiltonian is separable, we can rewrite the Hamiltonian in the form

$$\mathcal{H}(\boldsymbol{q}, \boldsymbol{p}) = T(\boldsymbol{p}) + V(\boldsymbol{q}). \tag{3}$$

This happens frequently in Hamiltonian mechanics, with $T$ being the kinetic energy and $V$ the potential energy. Substituting (3) into (1) yields

$$\begin{cases} \dfrac{\mathrm{d}\boldsymbol{q}}{\mathrm{d}t} = \dfrac{\partial T(\boldsymbol{p})}{\partial \boldsymbol{p}}, \\ \dfrac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}t} = -\dfrac{\partial V(\boldsymbol{q})}{\partial \boldsymbol{q}}. \end{cases} \tag{4}$$

This set of equations is fundamental in designing our neural networks. Our model will learn the right-hand side (r.h.s.) of (4) under the framework of ODE-net.

One of the important features of the time evolution of Hamilton's equations is symplectomorphism, which represents a transformation of phase space that is volume-preserving. In the setting of canonical coordinates, symplectomorphism means the transformation of the phase flow of a Hamiltonian system conserves the symplectic two-form

$$\mathrm{d}\boldsymbol{p} \wedge \mathrm{d}\boldsymbol{q} \equiv \sum_{j=1}^{N} \left(\mathrm{d}p_j \wedge \mathrm{d}q_j\right), \tag{5}$$

where $\wedge$ denotes the wedge product of two differential forms. Inspired by the symplectomorphism feature, we aim to construct a neural network architecture that intrinsically preserves Hamiltonian structure.

### 2.2 A symmetric network in Taylor expansion form

In order to learn the gradients of the Hamiltonian with respect to the generalized coordinates, we propose the following underpinning mechanism, which is a set of symmetric networks that learn the gradients of the Hamiltonian with respect to the generalized coordinates.

$$\begin{cases} \boldsymbol{T}_p(\boldsymbol{p}, \boldsymbol{\theta}_p) \to \dfrac{\partial T(\boldsymbol{p})}{\partial \boldsymbol{p}}, \\ \boldsymbol{V}_q(\boldsymbol{q}, \boldsymbol{\theta}_q) \to \dfrac{\partial V(\boldsymbol{q})}{\partial \boldsymbol{q}}, \end{cases} \tag{6}$$

with parameters $(\boldsymbol{\theta}_p, \boldsymbol{\theta}_q)$ that are designed to learn the r.h.s. of (4), respectively. Here, the "$\rightarrow$" represents our attempt to use the left-hand side (l.h.s) to learn the r.h.s. Substituting (6) into (4) yields

$$\begin{cases} \dfrac{\mathrm{d}\boldsymbol{q}}{\mathrm{d}t} = \boldsymbol{T}_p(\boldsymbol{p}, \boldsymbol{\theta}_p), \\ \dfrac{\mathrm{d}\boldsymbol{p}}{\mathrm{d}t} = -\boldsymbol{V}_q(\boldsymbol{q}, \boldsymbol{\theta}_q). \end{cases} \tag{7}$$

Therefore, under the initial condition (2), the trajectories of the canonical coordinates can be integrated as

$$\begin{cases} \boldsymbol{q}(t) = \boldsymbol{q}_0 + \displaystyle\int_{t_0}^{t} \boldsymbol{T}_p(\boldsymbol{p}, \boldsymbol{\theta}_p)\mathrm{d}t, \\ \boldsymbol{p}(t) = \boldsymbol{p}_0 - \displaystyle\int_{t_0}^{t} \boldsymbol{V}_q(\boldsymbol{q}, \boldsymbol{\theta}_q)\mathrm{d}t. \end{cases} \tag{8}$$

From (6), we obtain

$$\begin{cases} \dfrac{\partial \boldsymbol{T}_p(\boldsymbol{p}, \boldsymbol{\theta}_p)}{\partial \boldsymbol{p}} \rightarrow \dfrac{\partial^2 T(\boldsymbol{p})}{\partial \boldsymbol{p}^2}, \\ \dfrac{\partial \boldsymbol{V}_q(\boldsymbol{q}, \boldsymbol{\theta}_q)}{\partial \boldsymbol{q}} \rightarrow \dfrac{\partial^2 V(\boldsymbol{q})}{\partial \boldsymbol{q}^2}. \end{cases} \tag{9}$$

The r.h.s of (9) are the Hessian matrix of $T$ and $V$ respectively, so we can design $\boldsymbol{T}_p(\boldsymbol{p}, \boldsymbol{\theta}_p)$ and $\boldsymbol{V}_q(\boldsymbol{q}, \boldsymbol{\theta}_q)$ as symmetric mappings, that are

$$\frac{\partial \boldsymbol{T}_p(\boldsymbol{p}, \boldsymbol{\theta}_p)}{\partial \boldsymbol{p}} = \left[\frac{\partial \boldsymbol{T}_p(\boldsymbol{p}, \boldsymbol{\theta}_p)}{\partial \boldsymbol{p}}\right]^T, \tag{10}$$

and

$$\frac{\partial \boldsymbol{V}_q(\boldsymbol{q}, \boldsymbol{\theta}_q)}{\partial \boldsymbol{q}} = \left[\frac{\partial \boldsymbol{V}_q(\boldsymbol{q}, \boldsymbol{\theta}_q)}{\partial \boldsymbol{q}}\right]^T. \tag{11}$$

Due to the multiple nonlinear layers in the construction of traditional deep neural networks, it is impossible for these deep neural networks to fulfill (10) and (11). Therefore, we can only use a three-layer network with the form of *linear-activation-linear*, where the weights of the two linear layers are the transpose of each other, and in order to still maintain the expressive power of the networks, we construct symmetric nonlinear terms, as same as the terms of a Taylor polynomial, and combine them linearly. Specifically, we construct a symmetric network $\boldsymbol{T}_p(\boldsymbol{p}, \boldsymbol{\theta}_p)$ as

$$\boldsymbol{T}_p(\boldsymbol{p}, \boldsymbol{\theta}_p) = \left(\sum_{i=1}^{M} \boldsymbol{A}_i^T \circ f_i \circ \boldsymbol{A}_i - \boldsymbol{B}_i^T \circ f_i \circ \boldsymbol{B}_i\right) \circ \boldsymbol{p} + \boldsymbol{b}, \tag{12}$$

where '$\circ$' denotes the function composition, $\boldsymbol{A}_i$ and $\boldsymbol{B}_i$ are fully connected layers with size $N_h \times N$, $\boldsymbol{b}$ is a $N$ dimensional bias, $M$ is the number of terms in the Taylor series expansion, and $f_i$ is an element-wise function, representing the $i^{\text{th}}$ order term in the Taylor polynomial

$$f_i(x) = \frac{1}{i!}x^i. \tag{13}$$

Figure 1 plots a schematic diagram of $\boldsymbol{T}_p(\boldsymbol{p}, \boldsymbol{\theta}_p)$ in Taylor-net. The input of $\boldsymbol{T}_p(\boldsymbol{p}, \boldsymbol{\theta}_p)$ is $\boldsymbol{p}$, and $\boldsymbol{\theta}_p = (\boldsymbol{A}_i, \boldsymbol{B}_i, \boldsymbol{b})$. We construct a negative term $\boldsymbol{B}_i^T \circ f_i \circ \boldsymbol{B}_i$ following a positive term $\boldsymbol{A}_i^T \circ f_i \circ \boldsymbol{A}_i$, since two positive semidefinite matrices with opposite signs can represent any symmetric matrix.

To prove (12) is symmetric, that is it fulfills (10), we introduce theorem 2.1.

**Theorem 2.1.** *The network* (12) *satisfies* (10)*.*

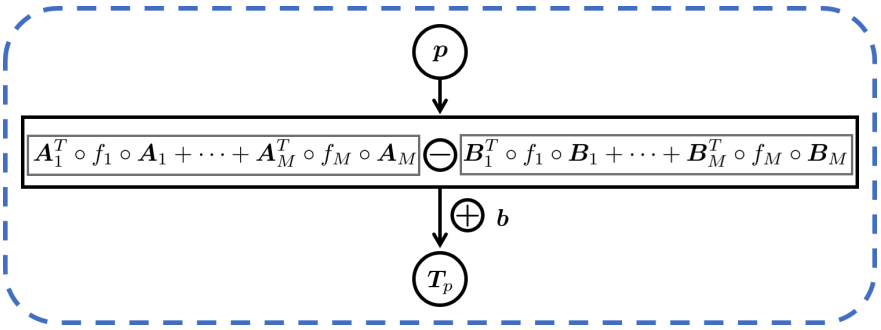


**Figure 1:** The schematic diagram of $\boldsymbol{T}_p(\boldsymbol{p}, \boldsymbol{\theta}_p)$ in Taylor-net.

*Proof.* From (12), we have

$$\frac{\partial \boldsymbol{T}_p(\boldsymbol{p}, \boldsymbol{\theta}_p)}{\partial \boldsymbol{p}} = \sum_{i=1}^{M} \boldsymbol{A}_i^T \boldsymbol{\Lambda}_i^A \boldsymbol{A}_i - \boldsymbol{B}_i^T \boldsymbol{\Lambda}_i^B \boldsymbol{B}_i, \tag{14}$$

with

$$\Lambda_i^A = \mathrm{diag}\left( \left. \frac{\mathrm{d}f}{\mathrm{d}x} \right|_{x = \boldsymbol{A}_i \circ \boldsymbol{p}} \right), \tag{15}$$

and

$$\Lambda_i^B = \mathrm{diag}\left( \left. \frac{\mathrm{d}f}{\mathrm{d}x} \right|_{x = \boldsymbol{B}_i \circ \boldsymbol{p}} \right). \tag{16}$$

It's easy to see that (14) is a symmetric matrix that satisfies (10). □

In fact, $\boldsymbol{T}_p(\boldsymbol{p}, \boldsymbol{\theta}_p)$ in (10) and $\boldsymbol{V}_q(\boldsymbol{q}, \boldsymbol{\theta}_q)$ in (11) satisfy the same property, so we construct $V_q$ with the similar form as

$$\boldsymbol{V}_q(\boldsymbol{q}, \boldsymbol{\theta}_q) = \left( \sum_{i=1}^{M} \boldsymbol{C}_i^T \circ f_i \circ \boldsymbol{C}_i - \boldsymbol{D}_i^T \circ f_i \circ \boldsymbol{D}_i \right) \circ \boldsymbol{q} + \boldsymbol{d}. \tag{17}$$

Here, $\boldsymbol{C}_i$, $\boldsymbol{D}_i$, and $\boldsymbol{d}$ have the same structure as (12), and $(\boldsymbol{C}_i, \boldsymbol{D}_i, \boldsymbol{d}) = \boldsymbol{\theta}_q$.

### 2.3 Symplectic Taylor neural networks

Next, we substitute the constructed network (12) and (17) into (8) to learn the Hamiltonian system (4). We employ ODE-net [5] as our computational infrastructure. Here we briefly introduce the essential idea of ODE-net for completeness. Under the perspective of viewing a neural network as a dynamic system, we can treat the chain of residual blocks in a neural network as the solution of an ODE with the Euler method. Given a residual network that consists of sequence of transformations

$$\boldsymbol{h}_{t+1} = \boldsymbol{h}_t + f(\boldsymbol{h}_t, \theta_t), \tag{18}$$

the idea is to parameterize the continuous dynamics using an ODE specified by a neural network:

$$\frac{\mathrm{d}\boldsymbol{h}(t)}{\mathrm{d}t} = f(\boldsymbol{h}_t, t, \theta). \tag{19}$$

Inspired by the idea of ODE-net, we design neural networks that can learn continuous time evolution. In Hamiltonian system (4), where the coordinates are integrated as (8), we can implement a time integrator to solve for $\boldsymbol{p}$ and $\boldsymbol{q}$. While

**Algorithm 1** Integrate (8) by using the fourth-order symplectic integrator

**Input:** $\boldsymbol{q}_0, \boldsymbol{p}_0, t_0, t, \Delta t$,
$\boldsymbol{F}_t^j$ in (20) and $\boldsymbol{F}_k^j$ in (21) with $j = 1, 2, 3, 4$;
**Output:** $\boldsymbol{q}(t), \boldsymbol{p}(t)$
$n = \text{floor}[(t - t_0)/\Delta t]$;
for $i = 1, n$
$(\boldsymbol{k}_p^0, \boldsymbol{k}_q^0) = (\boldsymbol{p}_{i-1}, \boldsymbol{q}_{i-1})$;
for $j = 1, 4$
$(\boldsymbol{t}_p^{j-1}, \boldsymbol{t}_q^{j-1}) = \boldsymbol{F}_t^j(\boldsymbol{k}_p^{j-1}, \boldsymbol{k}_q^{j-1}, \Delta t)$,
$(\boldsymbol{k}_p^j, \boldsymbol{k}_q^j) = \boldsymbol{F}_k^j(\boldsymbol{t}_p^{j-1}, \boldsymbol{t}_q^{j-1}, \Delta t)$,
end
$(\boldsymbol{p}_i, \boldsymbol{q}_i) = (\boldsymbol{k}_p^4, \boldsymbol{k}_q^4)$;
end
$\boldsymbol{q}(t) = \boldsymbol{q}_n, \boldsymbol{p}(t) = \boldsymbol{p}_n$.

ODE-net uses fourth-order Runge–Kutta method to make the neural networks structure-preserving, we need to implement an integrator that is symplectic. Therefore, we introduce Taylor-net, in which we design the symmetric Taylor series expansion and utilize the fourth-order symplectic integrator to construct neural networks that are symplectic to learn the gradients of the Hamiltonian with respect to the generalized coordinates and ultimately the temporal integral of a Hamiltonian system.

For the constructed networks (12) and (17), we integrate (8) by using the fourth-order symplectic integrator [12]. Specifically, we will have an input layer $(\boldsymbol{q}_0, \boldsymbol{p}_0)$ at $t = t_0$ and an output layer $(\boldsymbol{q}_n, \boldsymbol{p}_n)$ at $t = t_0 + n\mathrm{d}t$. The recursive relations of $(\boldsymbol{q}_i, \boldsymbol{p}_i), i = 1, 2, \cdots, n$, can be expressed by the algorithm 1. The input function in algorithm 1 are

$$\boldsymbol{F}_t^j(\boldsymbol{p}, \boldsymbol{q}, \mathrm{d}t) = (\boldsymbol{p}, \boldsymbol{q} + c_j \boldsymbol{T}_p(\boldsymbol{p}, \boldsymbol{\theta}_p)\mathrm{d}t)\,, \tag{20}$$

and

$$\boldsymbol{F}_k^j(\boldsymbol{p}, \boldsymbol{q}, \mathrm{d}t) = (\boldsymbol{p} - d_j \boldsymbol{V}_q(\boldsymbol{q}, \boldsymbol{\theta}_q)\mathrm{d}t, \boldsymbol{q})\,, \tag{21}$$

with

$$\begin{aligned} &c_1 = c_4 = \frac{1}{2(2 - 2^{1/3})}, \quad c_2 = c_3 = \frac{1 - 2^{1/3}}{2(2 - 2^{1/3})}, \\ &d_1 = d_3 = \frac{1}{2 - 2^{1/3}}, \qquad d_2 = -\frac{2^{1/3}}{2 - 2^{1/3}}, \qquad d_4 = 0. \end{aligned} \tag{22}$$

The derivation of the coefficients $c_j$ and $d_j$ can be found in [12, 50, 4]. Relationships (20) and (21) are obtained by replacing $\partial T(\boldsymbol{p})/\partial \boldsymbol{p}$ and $\partial V(\boldsymbol{q})/\partial \boldsymbol{q}$ in the fourth-order symplectic integrator with deliberately designed neural networks $\boldsymbol{T}_p(\boldsymbol{p}, \boldsymbol{\theta}_p)$ and $\boldsymbol{V}_q(\boldsymbol{q}, \boldsymbol{\theta}_q)$, respectively. Figure 2 plots a schematic diagram of Taylor-net which is described by algorithm 1. The input of Taylor-net is $(\boldsymbol{q}_0, \boldsymbol{p}_0)$, and the output is $(\boldsymbol{q}_n, \boldsymbol{p}_n)$. Taylor-net consists of $n$ iterations of fourth-order symplectic integrator. The input of the integrator is $(\boldsymbol{q}_{i-1}, \boldsymbol{p}_{i-1})$, and the output is $(\boldsymbol{q}_i, \boldsymbol{p}_i)$. Within the integrator, the output of $\boldsymbol{T}_p$ is used to calculate $\boldsymbol{q}$, while the output of $\boldsymbol{V}_q$ is used to calculate $\boldsymbol{p}$, which is signified by the shoelace-like pattern in the diagram. The four intermediate variables $\boldsymbol{t}_p^0 \cdots \boldsymbol{t}_p^4$ and $\boldsymbol{k}_q^0 \cdots \boldsymbol{k}_q^4$ indicate that the scheme is fourth-order.

By constructing the network $\boldsymbol{T}_p(\boldsymbol{p}, \boldsymbol{\theta}_p)$ in (12) that satisfies (10), we show that theorem 2.2 holds, so the network (20) preserves the symplectic structure of the system.

**Theorem 2.2.** *For a given dt, the mapping* $\boldsymbol{F}_t^j(:, :, dt) : \mathbb{R}^{2N} \to \mathbb{R}^{2N}$ *in* (20) *is a symplectomorphism if and only if the Jacobian of* $\boldsymbol{T}_p$ *is a symmetric matrix, that is, it satisifies* (10)*.*

*Proof.* Let

$$(\boldsymbol{t}_p, \boldsymbol{t}_q) = \boldsymbol{F}_t^j(\boldsymbol{k}_p, \boldsymbol{k}_q, \mathrm{d}t). \tag{23}$$

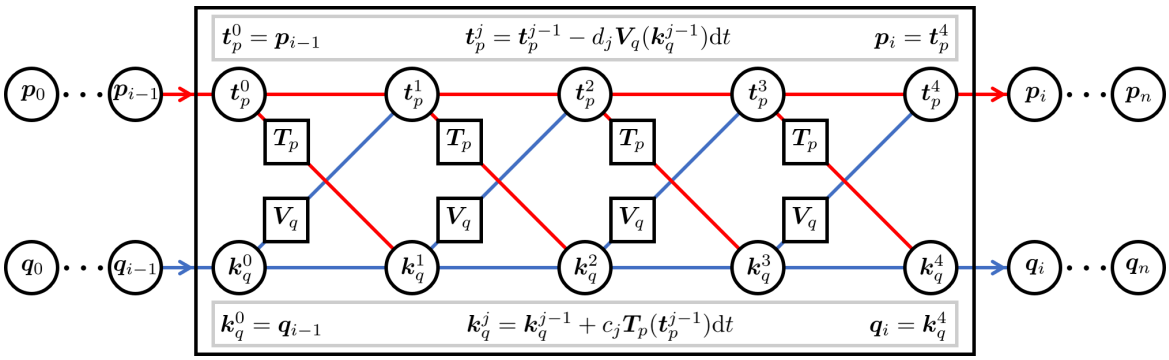


**Figure 2:** The schematic diagram of Taylor-net. The input of Taylor-net is $(\boldsymbol{q}_0, \boldsymbol{p}_0)$, and the output is $(\boldsymbol{q}_n, \boldsymbol{p}_n)$. Taylor-net consists of $n$ iterations of fourth-order symplectic integrator. The input of the integrator is $(\boldsymbol{q}_{i-1}, \boldsymbol{p}_{i-1})$, and the output is $(\boldsymbol{q}_i, \boldsymbol{p}_i)$. The four intermediate variables $\boldsymbol{t}_p^0 \cdots \boldsymbol{t}_p^4$ and $\boldsymbol{k}_q^0 \cdots \boldsymbol{k}_q^4$ show that the scheme is fourth-order.

From (20), we have

$$\begin{aligned} &\mathrm{d}\boldsymbol{t}_p \wedge \mathrm{d}\boldsymbol{t}_q = \mathrm{d}\boldsymbol{k}_p \wedge \mathrm{d}\boldsymbol{k}_q + \\ &\frac{1}{2}\sum_{l,m=1}^{N} c_j \mathrm{d}t \left[ \left.\frac{\partial \boldsymbol{T}_p(\boldsymbol{k}_p, \boldsymbol{\theta}_p)}{\partial \boldsymbol{k}_p}\right|_{l,m} - \left.\frac{\partial \boldsymbol{T}_p(\boldsymbol{k}_p, \boldsymbol{\theta}_p)}{\partial \boldsymbol{k}_p}\right|_{m,l} \right] \mathrm{d}\boldsymbol{k}_p|_l \wedge \mathrm{d}\boldsymbol{k}_q|_m. \end{aligned} \tag{24}$$

Here $\boldsymbol{A}|_{l,m}$ refers to the entry in the $l$-th row and $m$-th column of a matrix $\boldsymbol{A}$, $\boldsymbol{x}|_l$ refers to the $l$-th component of vector $\boldsymbol{x}$. From (24), we know that $\mathrm{d}\boldsymbol{t}_p \wedge \mathrm{d}\boldsymbol{t}_q = \mathrm{d}\boldsymbol{k}_p \wedge \mathrm{d}\boldsymbol{k}_q$ is equivalent to

$$\left.\frac{\partial \boldsymbol{T}_p(\boldsymbol{k}_p, \boldsymbol{\theta}_p)}{\partial \boldsymbol{k}_p}\right|_{l,m} - \left.\frac{\partial \boldsymbol{T}_p(\boldsymbol{k}_p, \boldsymbol{\theta}_p)}{\partial \boldsymbol{k}_p}\right|_{m,l} = 0, \quad \forall l, m = 1, 2, \cdots, N, \tag{25}$$

which is (10). □

Similar to the theorem2.2, we can find the relationship between $\boldsymbol{F}_k^j$ and the Jacobian of $\boldsymbol{V}_q$. The proof of 2.3 is omitted as it is similar to the proof of the theorem2.2.

**Theorem 2.3.** *For a given dt, the mapping* $\boldsymbol{F}_k^j(:,:,dt) : \mathbb{R}^{2N} \to \mathbb{R}^{2N}$ *in* (21) *is a symplectomorphism if and only if the Jacobian of* $\boldsymbol{V}_q$ *is a symmetric matrix, that is, it satisifies* (11).

Suppose that $\Phi_1$ and $\Phi_2$ are two symplectomorphisms. Then, it is easy to show that their composite map $\Phi_2 \circ \Phi_1$ is also symplectomorphism due to the chain rule. Thus, the symplectomorphism of the algorithm 1can be guaranteed by the theorems 2.2 and 2.3.

## 3 Numerical methods and results

This section discusses the details of our implementation, including the numerical method to generate training data, the construction of the neural networks, and the predictions for arbitrary time points on a continuous timeline.

### 3.1 Dataset Generation

To make a fair comparison with the ground truth, we generate our training and testing datasets by using the same numerical integrator based on a given analytical Hamiltonian. In the learning process, we generate $N_{train}$ training samples, and for each training sample, we first pick a random initial point $(\boldsymbol{q}_0, \boldsymbol{p}_0)$ (input), then use the symplectic integrator discussed in section 2.1 to calculate the value $(\boldsymbol{q}_n, \boldsymbol{p}_n)$ (target) of the trajectory at the end of the training period $T_{train}$. We do the same to generate a validation dataset with $N_{validation} = 100$ samples and the same time span as $T_{train}$ and calculate the validation loss $L_{validation}$ along the training loss $L_{train}$ to evaluate the training process. In addition, we generate a set of testing data with $N_{test} = 100$ samples and predicting time span $T_{predict}$ that is around

6000 times larger and calculate the prediction error $\epsilon_p$ to evaluate the predictive ability of the model. For simplicity, we use $(\hat{\boldsymbol{p}}_n, \hat{\boldsymbol{q}}_n)$ to represent the predicted values using our trained model.

We remark that our training dataset is relatively smaller than that used by the other methods. Most of the methods, e.g. ODE-net [5] and HNN [14], have to rely on intermediate data in their training data to train the model. That is the dataset is $[(\boldsymbol{q}_0^{(s)}, \boldsymbol{p}_0^{(s)}), (\boldsymbol{q}_1^{(s)}, \boldsymbol{p}_1^{(s)}), \ldots, (\boldsymbol{q}_{n-1}^{(s)}, \boldsymbol{p}_{n-1}^{(s)}), (\boldsymbol{q}_n^{(s)}, \boldsymbol{p}_n^{(s)})]_{s=1}^{N_{train}}$ , where $(\boldsymbol{q}_1^{(s)}, \boldsymbol{p}_1^{(s)}) \ldots, (\boldsymbol{q}_{n-1}^{(s)}, \boldsymbol{p}_{n-1}^{(s)})$ are $n-1$ intermediate points collected within $T_{train}$ in between $(\boldsymbol{q}_0^{(s)}, \boldsymbol{p}_0^{(s)})$ and $(\boldsymbol{q}_n^{(s)}, \boldsymbol{p}_n^{(s)})$. On the other hand, we only use two data points per sample, the initial data point and the end point, and our dataset looks like $\left[(\boldsymbol{q}_0^{(s)}, \boldsymbol{p}_0^{(s)}), (\boldsymbol{q}_n^{(s)}, \boldsymbol{p}_n^{(s)})\right]_{s=1}^{N_{train}}$, which is $n-1$ times smaller the dataset of the other methods, if we do not count $(\boldsymbol{q}_0^{(s)}, \boldsymbol{p}_0^{(s)})$. Our predicting time span $T_{predict}$ is around 6000 times the training period used in the training dataset $T_{train}$ (as compared to 10 times in HNN). This leads to a 600 times compression of the training data, in the dimension of temporal evolution. Note that we fix $T_{train}$ and $T_{predict}$ in practice so that we can train our network more efficiently on GPU. One can also choose to generate training data with different $T_{train}$ for each sample to obtain more robust performance.

### 3.2 Test Cases

We consider the pendulum, the Lotka–Volterra, the Kepler, and the Hénon–Heiles systems in our implementation.

**Pendulum system** The Hamiltonian of an ideal pendulum system is given by

$$\mathcal{H}(q, p) = \frac{1}{2}p^2 - \cos(q). \tag{26}$$

We pick a random initial point for training $(\boldsymbol{q}_0, \boldsymbol{p}_0) \in [-2, 2] \times [-2, 2]$.

**Lotka–Volterra system** For a Lotka–Volterra system, its Hamiltonian is given by

$$\mathcal{H}(q, p) = p - e^p + 2q - e^q. \tag{27}$$

Similarly, we pick a random initial point for training $(\boldsymbol{q}_0, \boldsymbol{p}_0) \in [-2, 2] \times [-2, 2]$.

**Kepler system** Now we consider a eight-dimensional system, a two-body problem in 2-dimensional space. Its Hamiltonian is given by

$$\mathcal{H}(\boldsymbol{q}, \boldsymbol{p}) = \mathcal{H}(q_1, q_2, q_3, q_4, p_1, p_2, p_3, p_4) = \frac{1}{2}(p_1^2 + p_2^2 + p_3^2 + p_4^2) - \frac{1}{\sqrt{q_1^2 + q_2^2 + q_3^2 + q_4^2}}, \tag{28}$$

where $(q_1, q_2)$ and $(p_1, p_2)$ are the position and momentum associated with the first body, $(q_1, q_2)$ and $(p_3, p_4)$ are the position and momentum associated with the second body. We randomly pick the initial training point $(\boldsymbol{q}_0, \boldsymbol{p}_0) \in [-3, 3] \times [-2, 2]$, and enforce a constraint on the initial $(q_1, q_2)$ and $(p_1, p_2)$ so that they are at least separated by some distance $L_d = 4$. This is to avoid having infinite force immediately.

**Hénon–Heiles system** Lastly, we introduce a four-dimensional Hénon–Heiles system, which is a non-integrable system. This kind of chaotic system is generally hard to model. Its Hamiltonian is defined as

$$\mathcal{H}(\boldsymbol{q}, \boldsymbol{p}) = \mathcal{H}(q_1, q_2, p_1, p_2) = \frac{1}{2}(p_1^2 + p_2^2) + \frac{1}{2}(q_1^2 + q_2^2) + (q_1^2 q_2 - \frac{q_2^3}{3}), \tag{29}$$

The random initial point for training is $(\boldsymbol{q}_0, \boldsymbol{p}_0) \in [-0.5, 0.5] \times [-0.5, 0.5]$.

### 3.3 Training settings and ablation tests

For all four systems, we use the Adam optimizer [25]. We choose the automatic differentiation method as our backward propagation method. We have tried both the adjoint sensitivity method, which is used in ODE-net [5] and the automatic differentiation method. Both methods can be used to train the model well. However, we found that using the adjoint sensitivity method is much slower than using the automatic differentiation method considering the large parameter size of neural networks. Therefore, we use the automatic differentiation method in our implementation. The detailed derivation of adjoints formulas under the setting of Taylor-net and the prediction result can be found in A.

**Table 1**
*Set-up of problems.*

| Problems | Pendulum | Lotka-Volterra | Kepler | Hénon–Heiles |
|---|---|---|---|---|
| Hamiltonian | (26) | (27) | (28) | (29) |
| $T_{train}$ | 0.01 | 0.01 | 0.01 | 0.01 |
| $T_{predict}$ | $20\pi$ | $20\pi$ | $20\pi$ | 10 |
| $N_{train}$ | 15 | 25 | 25 | 25 |
| Epoch | 100 | 150 | 50 | 100 |
| Learning rate | 0.002 | 0.003 | 0.001 | 0.001 |
| $step\_size$ | 10 | 10 | 10 | 10 |
| $\gamma$ | 0.8 | 0.8 | 0.8 | 0.8 |
| $M$ | 8 | 8 | 20 | 12 |
| $N_h$ | 16 | 8 | 8 | 16 |
| $L_{train}$ | $2.75 \times 10^{-5}$ | $2.37 \times 10^{-5}$ | $7.29 \times 10^{-5}$ | $9.24 \times 10^{-6}$ |
| $L_{validation}$ | $1.39 \times 10^{-4}$ | $6.73 \times 10^{-5}$ | $6.41 \times 10^{-5}$ | $9.44 \times 10^{-6}$ |

All $A_i$ and $B_i$ in (12) are initialized as $A_i, B_i \sim \mathcal{N}(0, \sqrt{2/[N * N_h * (i+1)]})$, where $N$ is the dimension of the system and $N_h$ is the size of the hidden layers. The loss function is

$$L_{train} = \frac{1}{N_{train}} \sum_{s=1}^{N_{train}} \|\hat{\boldsymbol{p}}_n^{(s)} - \boldsymbol{p}_n^{(s)}\|_1 + \|\hat{\boldsymbol{q}}_n^{(s)} - \boldsymbol{q}_n^{(s)}\|_1. \tag{30}$$

The validation loss $L_{validation}$ is the same as (30) but with dataset different from the training dataset. We choose $L1$ loss, instead of Mean Square Error (MSE) loss because $L1$ loss performs better in all cases given in Table 1. We conduct the ablation test on these problems to compare the validation loss after convergence with different training loss functions in the training process. Figure 14 shows the comparison of validation losses with different training loss functions in the training process of different problems validated by L1 loss function. Figure 15 shows the comparison of validation losses with different training loss functions in the training process of different problems validated by MSE loss function. We observe that for all problems, the validation loss with $L1$ is smaller than that with MSE after convergence. We believe the better performance of $L1$ may be due to MSE loss's high sensitivity to outliers. Hence, we choose to use $L1$ loss as our training loss function.

The details of the parameters we set and some other important quantities can be found in Table 1. To show the predictive ability of our model, we pick $T_{predict} = 20\pi$ for the pendulum, the Lotka–Volterra and the Kepler problems. For the Hénon–Heiles problem, we pick $T_{predict} = 10$ because of its chaotic nature. We pick 15 as the sample size for the pendulum problem and 25 for other problems since we find that small $N_{train}$'s are sufficient to generate excellent results. More discussions about $N_{train}$ can be found in section 3.6. The epoch parameter represents the number of epochs needed for the training loss to converge. $step\_size$ indicates the period of learning rate decay, and $\gamma$ is the multiplicative factor of learning rate decay. These two parameters decay the learning rate of each parameter group by $\gamma$ every $step\_size$ epochs, which prevents the model from overshooting the local minimum. The dynamic learning rate can also make our model converge faster. $M$ indicates the number of terms of the Taylor polynomial introduced in the construction of the neural networks (12). Through experimentation, we find that 8 terms can represent most functions well. Therefore, we pick $M = 8$ for the pendulum and the Lotka-Volterra problems. For more complicated systems, like the Kepler and the Hénon–Heiles systems, we choose $M = 20$ and $M = 12$, respectively.

$N_h$, the dimension of hidden layers, is a parameter that needs to be carefully chosen. We conduct the ablation test on the pendulum, the Lotka–Volterra, the Kepler, and the Hénon–Heiles problems to compare the validation loss using different $N_h$. Figure 3 shows the results of the test. From figure 3(a), it can be seen that the validation loss after convergence for the pendulum problem drops significantly after increasing $N_h$ from 8 to 16 and then stays relatively similar with higher $N_h$. Therefore, we choose to use 16 as $N_h$ for the pendulum problem. Following the same logic, we choose 8, 8, and 16 as $N_h$ for the Lotka–Volterra, the Kepler, and the Hénon–Heiles problems. Notice that $N_h$ for the lower-dimensional problem, namely, the pendulum problem, is larger than $N_h$ for the higher dimensional problem, the Kepler problem. This is because, for the higher-dimensional problem, the degree of freedom is actually more limited. This is due to the prior knowledge that the forces between objects are the same.

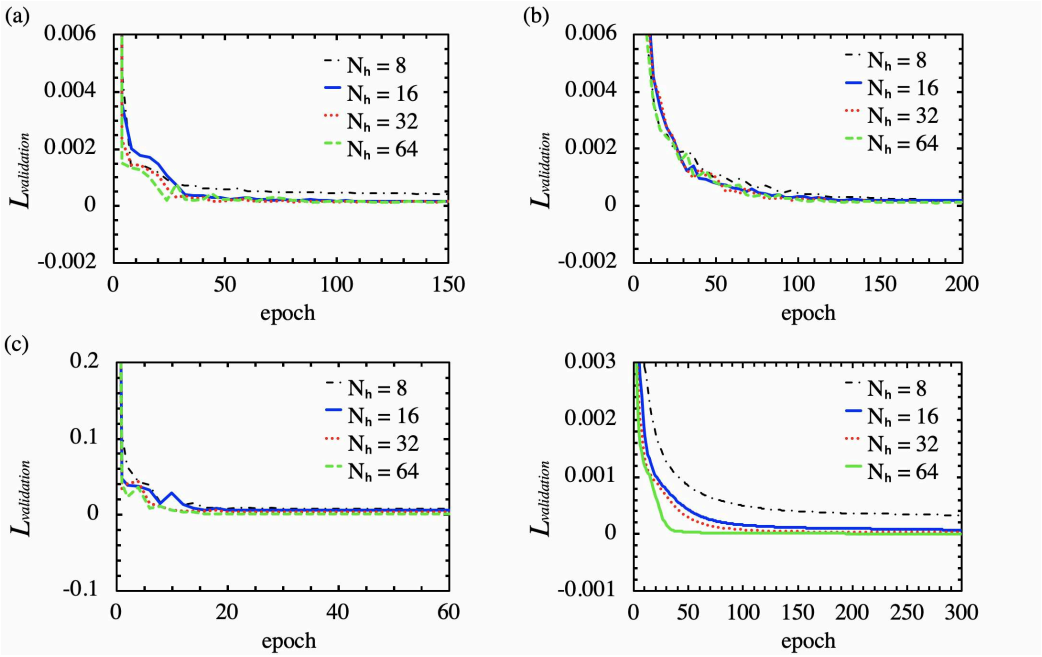


**Figure 3:** Comparisons of validation losses with different $N_h$ in the training process for (a) the pendulum, (b) the Lotka–Volterra, (c) the Kepler, and (d) the Hénon–Heiles problems.

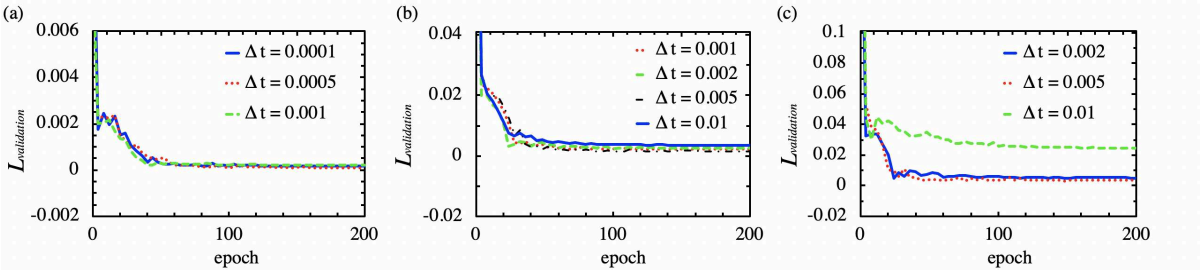


**Figure 4:** Comparisons of validation losses with different $dt$ in the training process. (a), (b), and (c) are trained based on different time spans $T_{train} = 0.01$, $0.1$, and $0.2$, respectively.

Another vital parameter that is not mentioned in Table 1 is the integral time step $\Delta t$ in the sympletic integrator. Notice that the choice of $\Delta t$ largely depends on the time span $T_{train}$. Figure 4 compares the validation losses generated by various integral time steps $\Delta t$ based on fixed dataset time spans $T_{train} = 0.01$, $0.1$ and $0.2$ respectively in the training process. For the concern of gradient vanishing or exploding, notice that when the number of iterations $n$ is big, which is when $\Delta t$ is small, we did not observe these issues, as shown in 4(a), where the smallest $\Delta t$ is $10^{-4}$. Since we embed the structure of residual networks in our symplectic integrator, there should not be the problem of vanishing gradient. It is clear that the validation loss converges to a similar degree with various $\Delta t$ based on fixed $T_{train} = 0.01$ and $T_{train} = 0.1$ in 4(a) and (b), while it increases significantly as $\Delta t$ increases based on fixed $T_{train} = 0.02$ in 4(c). Thus, we need to be careful when choosing $n$, or $\Delta t$, for the dataset with larger time span $T_{train}$.

We record the training loss for all the problems at the epochs specified above. It is worth noticing that the training loss of our model is at $10^{-5}$ order of magnitude and below, which indicates our model's ability to fit the training data. As we can see from figure 5, the prediction results using Taylor-net match perfectly with the ground truth for all three systems, even though the $T_{train} = 0.01$ is $2000\pi$ times shorter than the $T_{predict} = 20\pi$ in figure 5 (a) and (b), and 1000 times shorter in figure 5 (c). In particular, our model predicts the dynamics of the chaotic system, the Hénon–Heiles system (29) extremely well, which regular neural networks fail to do. The results indicate the compelling predictive ability of our model. This can be seen more clearly in 3.5 when we compare Taylor-net with other methods.

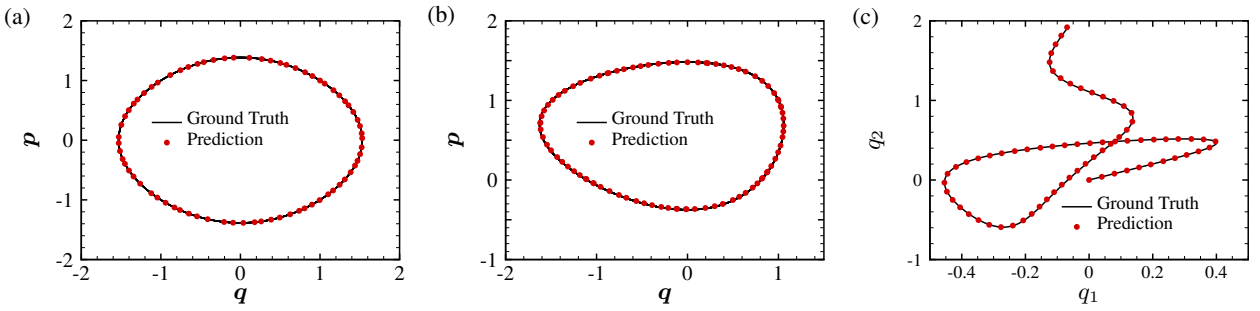


**Figure 5:** Prediction result using Taylor-net for (a) the pendulum, (b) the Lotka–Volterra, and (c) the Hénon–Heiles problems. For better visualization, we set the initial points as (a) $(\boldsymbol{q}_0, \boldsymbol{p}_0) = (1, 1)$, (b) $(\boldsymbol{q}_0, \boldsymbol{p}_0) = (1, 1)$, and (c) $(\boldsymbol{q}_0, \boldsymbol{p}_0) = ([0, 0], [0.5, 0.5])$. The prediction results using Taylor-net match perfectly with the ground truth for all three systems, even though the $T_{train}$ is $2000\pi$ times shorter than the $T_{predict}$ in (a) and (b), and 1000 times shorter in (c). $T_{train} = 0.01$ and $T_{predict} = 20\pi$ in (a) and (b), and $T_{train} = 0.01$ and $T_{predict} = 10$ in (c).

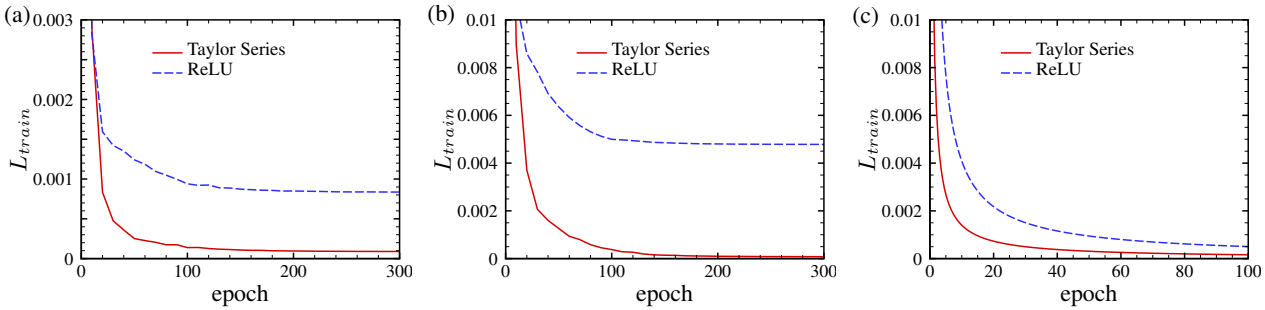


**Figure 6:** Mean of $L_{train}$ using Taylor series vs. using ReLU for (a) the pendulum, (b) the Lotka–Volterra, and (c) the Kepler problems. We train each model until $L_{train}$ converges and average $L_{train}$ for (a) every 10 epochs for the pendulum problem, (b) every 10 epochs for the Lotka–Volterra problem, and (c) every 5 epochs for the Kepler problem.

### 3.4 Taylor series vs. ReLU

In order to evaluate the performance of using Taylor series as the underlying structure of Taylor-net to ensure non-linearity, we also implement the most commonly used activation function, ReLU and compare the training loss with our current model. We construct the neural networks as (12) with parameters specified in Table 1, except we use $f_i(x) = \max(0, x)$ instead. The experimental results show that the neural networks perform better with Taylor series than with ReLU in the pendulum, the Lotka–Volterra, and the Kepler problems. We can observe from figure 6 that in all three problems the loss of using ReLU is larger than the loss of using Taylor series after the loss converges. In the pendulum problem, the mean of loss after convergence from 100 epochs to 300 epochs using the Taylor series is $8.878 \times 10^{-5}$, while that of using ReLU is $8.348 \times 10^{-4}$, which is 10 times larger than the mean of loss using Taylor series. The difference in the Lotka–Volterra problem is even more obvious. The mean of loss from 100 epochs to 300 epochs using the Taylor series is $7.832 \times 10^{-5}$, while that of using ReLU is $4.782 \times 10^{-3}$. In the Kepler problem, the mean of loss from 40 epochs to 100 epochs using the Taylor series is $2.524 \times 10^{-4}$, while that of using ReLU is $8.408 \times 10^{-4}$. In all three problems, the Taylor series performs undoubtedly better than ReLU. Thus, the results clearly show that using the Taylor series gives a better approximation of the dynamics of the system. The strong representational ability of the Taylor series is an important factor that increases the accuracy of the prediction.

### 3.5 Predictive ability and robustness

Now, to assess how well our method can predict the future flow, we compare the predictive ability of Taylor-net with ODE-net and HNN. We apply all three methods on the pendulum problem, and let $T_{train} = 0.01$ and $T_{predict} = 20\pi$. We evaluate the performance of the models by calculating the average prediction error at each predicted points, defined

**Table 2**
*Comparison of $\epsilon_p$ for the pendulum problem without noise, with noise $\sigma_1, \sigma_2 \sim \mathcal{N}(0, 0.1)$, and with noise $\sigma_1, \sigma_2 \sim \mathcal{N}(0, 0.5)$.*

| Methods | Taylor-net | HNN | ODE-net |
|---|---|---|---|
| $\epsilon_p$, without noise | 0.213 | 0.377 | 1.416 |
| $\epsilon_p$, with noise $\sigma_1, \sigma_2 \sim \mathcal{N}(0, 0.1)$ | 1.667 | 2.433 | 3.301 |
| $\epsilon_p$, with noise $\sigma_1, \sigma_2 \sim \mathcal{N}(0, 0.5)$ | 1.293 | 2.416 | 27.114 |

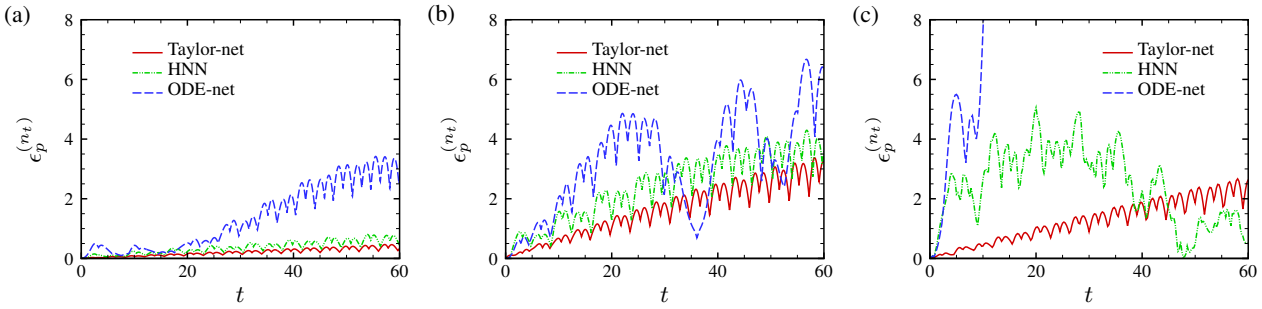


**Figure 7:** Prediction error $\epsilon_p^{(n_t)}$ at different $t$ from $t = 0$ to $t = 20\pi$ for the pendulum problem (a) without noise, (b) with noise $\sigma_1, \sigma_2 \sim \mathcal{N}(0, 0.1)$, and (c) with noise $\sigma_1, \sigma_2 \sim \mathcal{N}(0, 0.5)$. In the figure, $t = n_t \Delta t$, where $\Delta t = 0.01$. $\epsilon_p^{(n_t)}$ is the prediction error at the $n_t^{\text{th}}$ predicted point among the total $N_T = T_{predict}/\Delta t$ predicted points. We use $T_{train} = 0.01$, $T_{train} = 0.5$ and $T_{train} = 1$ to train the model in (a), (b), and (c), respectively.

by

$$\epsilon_p^{(n_t)} = \frac{1}{N_{test}} \sum_{s=1}^{N_{test}} \|\hat{\boldsymbol{p}}_n^{(s,n_t)} - \boldsymbol{p}_n^{(s,n_t)}\|_1 + \|\hat{\boldsymbol{q}}_n^{(s,n_t)} - \boldsymbol{q}_n^{(s,n_t)}\|_1, \tag{31}$$

and the average $\epsilon_p^{(n_t)}$ over $T_{predict}$ is

$$\epsilon_p = \frac{1}{N_T} \sum_{n_t=1}^{N_T} \epsilon_p^{(n_t)}, \tag{32}$$

where $N_{test}$ represents the testing sample size specified in section 3.1 and $N_T = T_{predict}/\Delta t$ with $\Delta t = 0.01$. After experimentation, we find that Taylor-net has stronger predictive ability than the other two methods. The first row of Table 2 shows the average prediction error of 100 testing samples using the three methods over $T_{predict}$ when no noise is added. The prediction error of HNN is almost double that of Taylor-net, while the prediction error of ODE-net is about 7 times that of Taylor-net. To analyze the difference more quantitatively, we made several plots to help us better compare the prediction results. Figure 7 shows the plots of prediction error $\epsilon_p^{(n_t)}$ against $t = n_t \Delta t$ over $T_{predict}$ for all three methods. In figure 8, we plot the prediction of position $q$ against time period for all three methods as well as the ground truth in order to see how well the prediction results match the ground truth. From figure 8 (a), we can already see that the prediction result of ODE-net gradually deviates from the ground truth as time progresses, while the prediction of Taylor-net and HNN stays mostly consistent with the ground truth, with the former being slightly closer to the ground truth. The difference between Taylor-net and HNN can be seen more clearly in figure 7 (a). Observe that the prediction error of Taylor-net is obviously smaller than that of the other two methods, and the difference becomes more and more apparent as time increases. The prediction error of ODE-net is larger than HNN and Taylor-net at the beginning of $T_{predict}$ and increases at a much faster rate than the other two methods. Although the prediction error of HNN has no obvious difference from that of Taylor-net at the beginning, it gradually diverges from the prediction error of Taylor-net.

Additionally, in figure 9, we plot the numerically solved ground truth, Taylor-net, HNN, and ODE-net calculated Hamiltonian for the pendulum problem. Figure 9 shows that the Taylor-net preserves the Hamiltonian relatively successfully, while ODE-net diverges away from the ground truth quickly. Although the predicting result of HNN does not seem to drift away from the ground truth, the divergent amplitude of HNN is greater than that of Taylor-net. Note that our model strictly preserves the symplectic structure, which is a geometric structure that cannot be

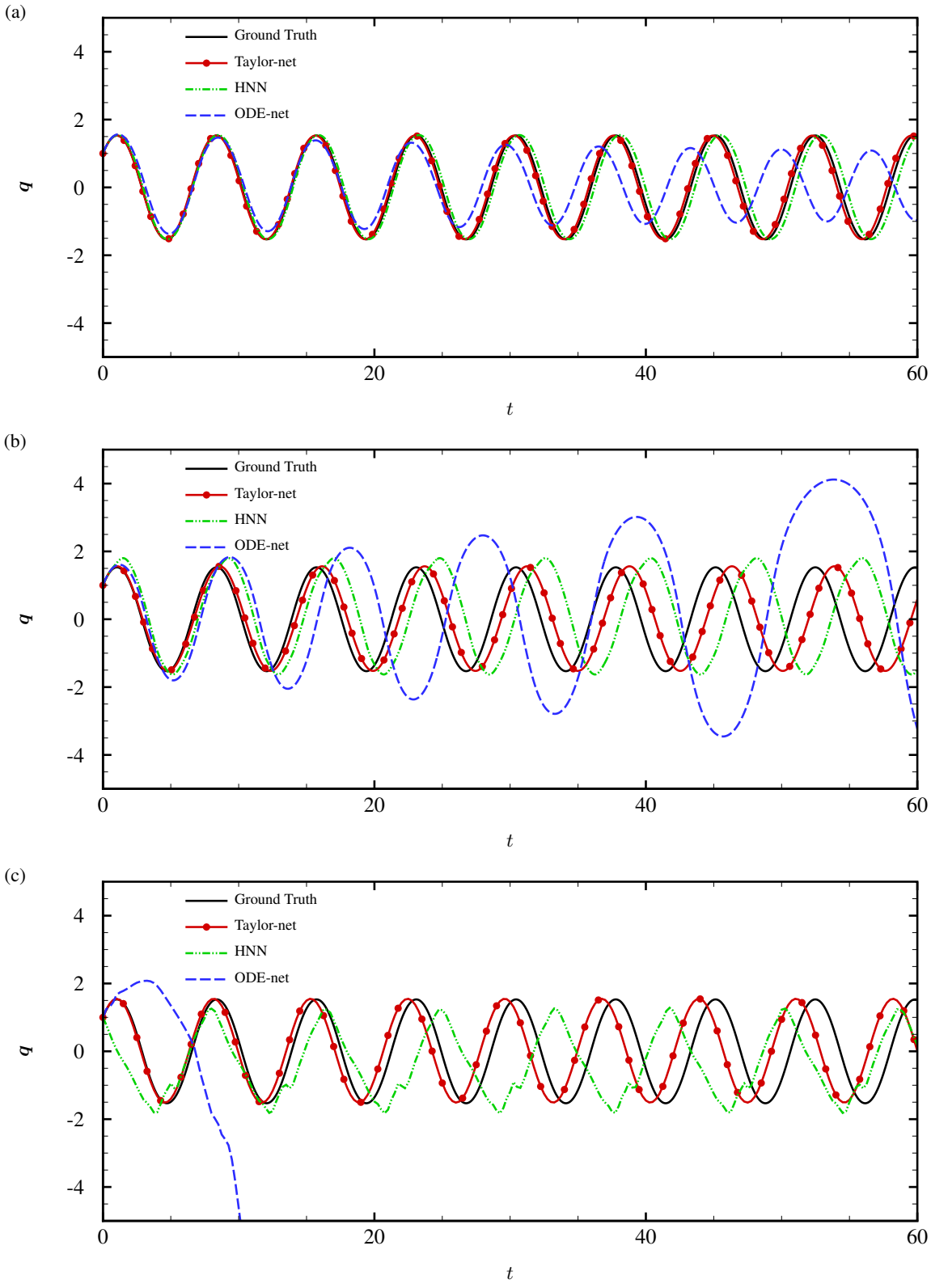


**Figure 8:** Prediction results of position $\boldsymbol{q}$ from $t = 0$ to $t = 20\pi$ for the pendulum problem using Taylor-net, HNN, and ODE-net (a) without noise, (b) with noise $\sigma_1, \sigma_2 \sim \mathcal{N}(0, 0.1)$, and (c) with noise $\sigma_1, \sigma_2 \sim \mathcal{N}(0, 0.5)$. For all the models, we set the initial point as $(\boldsymbol{q}_0, \boldsymbol{p}_0) = (1, 1)$. We use $T_{train} = 0.01$, $T_{train} = 0.5$ and $T_{train} = 1$ to train the model in (a), (b), and (c), respectively. All the methods are trained until the $L_{validation}$ converges.

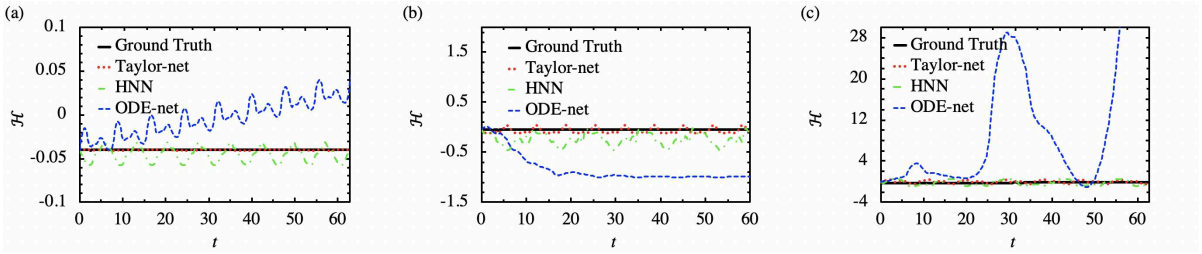


**Figure 9:** Prediction results of Hamiltonian $\mathcal{H}$ from $t = 0$ to $t = 12\pi$ for the pendulum problem (a) without noise, (b) with noise $\sigma_1, \sigma_2 \sim \mathcal{N}(0, 0.1)$, and (c) with noise $\sigma_1, \sigma_2 \sim \mathcal{N}(0, 0.5)$.

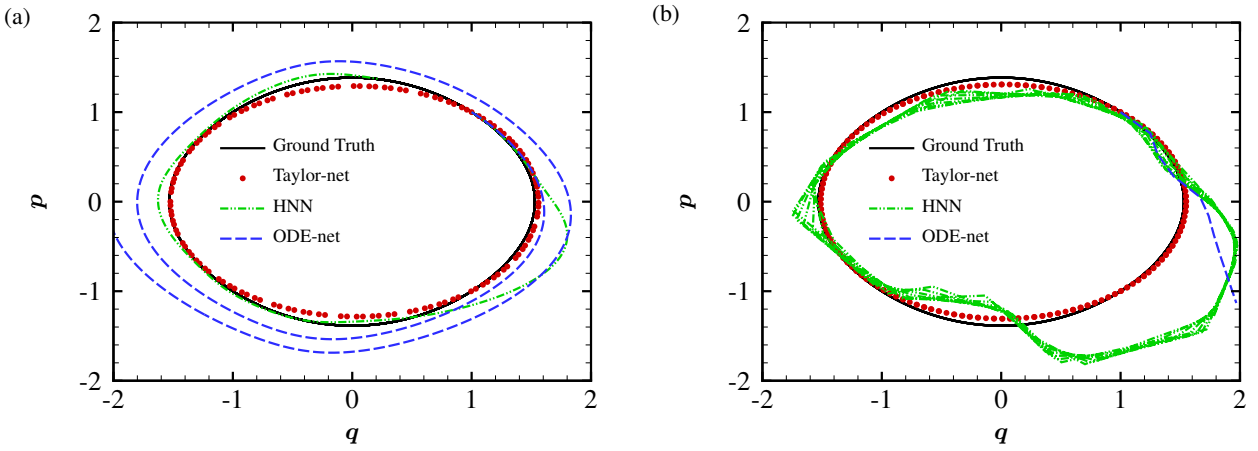


**Figure 10:** Prediction results of position $\boldsymbol{q}$ and momentum $\boldsymbol{p}$ from $t = 0$ to $t = 20\pi$. (a) with noise $\sigma_1, \sigma_2 \sim \mathcal{N}(0, 0.1)$ and (b) with noise $\sigma_1, \sigma_2 \sim \mathcal{N}(0, 0.5)$ in training process. We use $T_{train} = 0.5$ and $T_{train} = 1$ to train the model in (a) and (b) respectively. All the methods are trained until the $L_{validation}$ converges. In (a), we only plot the result of ODE-net until $t = 4\pi$ because the result beyond that will further diverge from the ground truth and cannot be fit into the graph. For the same reason, we only plot the result of ODE-net until $t = \pi$ in (b).

quantitatively calculated and plotted. Since the symplectic structure is preserved, the Hamiltonian predicted by our model is much closer to the ground truth.

In real systems, it is almost impossible to collect data without noise. Therefore, with noisy data, the robustness of neural networks is particular important. Instead of using $(\boldsymbol{q}_n, \boldsymbol{p}_n)$ to train the model, we add some random noise to the true value so that it becomes $(\boldsymbol{q}_n + \sigma_1, \boldsymbol{p}_n + \sigma_2)$. We test three models on two cases with small and large noises. We add noise $\sigma_1, \sigma_2 \sim \mathcal{N}(0, 0.1)$ in the case of small noise and $\sigma_1, \sigma_2 \sim \mathcal{N}(0, 0.5)$ in the case of large noise. We use $T_{train} = 0.5$ and $T_{train} = 1$ to train the model in the cases of small and large noises respectively. In both cases, we use 50 samples and make prediction over $T_{predict} = 20\pi$.

Figure 10 shows the predicted $\boldsymbol{p}$ versus $\boldsymbol{q}$ using different methods. From figure 10 (a), we find that Taylor-net discovers the unknown trajectory successfully, while ODE-net diverges from the true value quickly. Although the predicting result of HNN does not seem to drift away from the true dynamics, it does not fit the true trajectory as well as the prediction made by Taylor-net. The difference becomes clearer as we increase the noise. From figure 10 (b), we observe that Taylor-net still makes predictions that are almost consistent with the true trajectories, while ODE-net completely fails to do so. Moreover, the prediction made by HNN is much worse than in the case of small noise, while the performance of Taylor-net remains as good as the previous case.

This can be more clearly seen from figure 7 (b) and (c). We can see that $\epsilon_p^{(n_t)}$ of Taylor-net is consistent in both cases of small and large noises, while $\epsilon_p^{(n_t)}$ of HNN and ODE-net increase significantly and exhibit more fluctuation. It is worth noticing that in figure 7 (c), $\epsilon_p^{(n_t)}$ of HNN becomes smaller towards the end of $T_{predict}$. However, it is not because the performance of HNN becomes better, but rather due to the fact that the predicted flow of HNN is off by one period of motion, which can be seen from figure 8 (c). The second and third rows of Table 2 also give an overview on how the prediction error $\epsilon_p$ over $T_{predict}$ of the three methods differ. From figure 8 (b) and (c), we can clearly

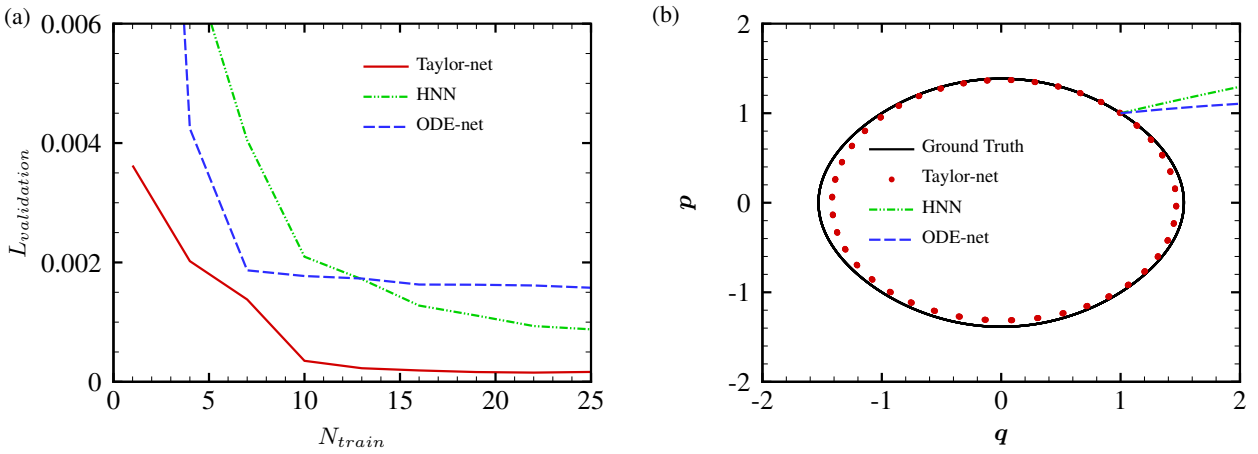


**Figure 11:** (a) At 100 epochs, $L_{validation}$ as a function of sample size ranging from $N_{train} = 1$ to $N_{train} = 25$. The $L_{validation}$ is averaged over 50 trials. (b) Prediction results of position $\boldsymbol{q}$ and momentum $\boldsymbol{p}$ from $t = 0$ to $t = 2\pi$ from $t = 0$ to $t = 20\pi$ using trained models after 1 epoch.

observe that the amplitude of predicted $\boldsymbol{q}$ using ODE-net increases as $t$ increases, and the amplitude of predicted $\boldsymbol{q}$ using HNN is slightly larger or smaller than that of the ground truth from the beginning. In contrast, due to the intrinsic symplectic structure of Taylor-net, the amplitude of predicted $\boldsymbol{q}$ using Taylor-net is inconsistent with the ground truth, without changing in time. Additionally, it is obvious that the predicted $\boldsymbol{q}$ using Taylor-net has the smallest phase shift among the three methods.

### 3.6 Training sample size and convergence rate

Besides the strong predictive ability and robustness, we also want to highlight the significantly small $N_{train}$ and the fast convergence rate of our approach. In a complex physical system, the cost of acquiring data is high. Our model can learn from the dataset that contains less than 15 samples and still generate validation loss $L_{validation}$ that is below $10^{-4}$. In figure 11 (a), we plot the $L_{validation}$ as a function of sample size using Taylor-net, HNN, and ODE-net. To make a fair comparison, we average the values of $L_{validation}$ over 50 trials. We can observe that the $L_{validation}$ for Taylor-net at 1 sample is around 5 times smaller than the $L_{validation}$ for ODE-net and the $L_{validation}$ for HNN. Although there are some fluctuations in $L_{validation}$ due to small $N_{train}$, the $L_{validation}$ for Taylor-net converges at around 10 samples, while the $L_{validation}$ for HNN is still decreasing. Although the $L_{validation}$ for ODE-net also converges around 10 samples, the value of its $L_{validation}$ is 10 times larger than that the $L_{validation}$ for Taylor-net.

Because of the intrinsically structure-preserving nature of our model, our model can well predict the dynamics of the underlying system even when it is trained for only a few epochs. In figure 11 (b), we plot the prediction results from $t = 0$ to $t = 20\pi$ using Taylor-net, HNN, and ODE-net after only 1 epoch of training. The prediction results made by HNN and ODE-net completely fail to match the true flow, while Taylor-net predicts the truth to a level that can never be achieved using HNN and ODE-net at such a small number of epochs.

We summarize the main traits and performance of the three methodologies in Table 3. We already emphasized enough that our model utilizes physics prior through constructing neural networks that intrinsically preserve the symplectic structure. Due to our model's structure-preserving ability, it can make accurate predictions with a very small training dataset that does not require any intermediate data. We also want to mention that HNN and ODE-net both require the analytical solutions of the temporal derivatives to train their models, which are often not obtainable from real systems. Moreover, besides the qualitative differences, we also compare the three methods quantitatively. In the pendulum problem, we fix the sample size to be 15 and find Taylor-net only needs 100 epochs for $L_{train}$ to converge, while HNN and ODE-net need 1000 epochs and 7000 epochs respectively. We also test how many samples Taylor-net, HNN, and ODE-net need for $L_{validation}$ to decrease to $10^{-4}$. Notice that we train Taylor-net, HNN, and ODE-net until convergence, which is for 100, 1000, and 7000 epochs respectively. Taylor-net only needs 15 samples and 100 epochs of training to achieve $L_{validation} \sim 10^{-4}$, while HNN needs 50 samples and 1000 epochs and ODE-net needs 50 samples and 7000 epochs. If we train HNN and ODE-net for 100 epochs in the same manner as Taylor-net, their $L_{validation}$ will never reach $10^{-4}$.

**Table 3**
*Comparison between Taylor-net, HNN, ODE-net. ✓ represents the method preserves such property.*

| Methods | Taylor-net | HNN | ODE-net |
|---|---|---|---|
| Utilize physics prior | ✓ | ✓ | Partially |
| Preserve symplectic structure | ✓ | Partially | |
| No need for intermediate training data | ✓ | | |
| No need for analytical solution of derivative | ✓ | | |
| Number of epochs until $L_{train}$ converges* | 100 | 1000 | 7000 |
| Sample size needed for $L_{validation} \sim 10^{-4}$ # | 15 | 50 | 50 |

*In the pendulum problem with sample size 15

#In the pendulum problem, train each model until convergence

## 4 High-dimensional systems

We want to extend our model into higher-dimensional dynamical systems. Let's consider a more complicated system, a multidimensional N-body system. Its Hamiltonian is given by

$$\mathcal{H}(\boldsymbol{q},\boldsymbol{p}) = \frac{1}{2}\sum_{i=1}^{N_{body}} \|\boldsymbol{p_i}\|^2 - \sum_{1\le i<j\le N_{body}} \frac{1}{\|\boldsymbol{q_j}-\boldsymbol{q_i}\|}, \tag{33}$$

where $N_{body}$ is the number of bodies in the system, and $N_{body} \times (\text{dimension of space}) = N$.

In a two-dimensional space, consider a system with $N_{body} > 2$ bodies. The cost of collecting training data from all $N_{body}$ bodies may be high, and the training process may be time inefficient. Thus, instead of collecting information from all $N_{body}$ bodies to train our model, we only use data collected from two bodies as training data to make a prediction of the dynamics of $N_{body}$ bodies. This is based on the assumption that the interactive models between particle pairs with unit particle strengths $m = 1$ are the same, and their corresponding Hamiltonian can be represented as network $\hat{\mathcal{H}}_\theta(\boldsymbol{x}_j, \boldsymbol{x}_k)$, based on which the corresponding Hamiltonian of $N_{body}$ particles can be written as [1, 40]

$$\mathcal{H}_\theta = \sum_{i,j=1}^{N_{body}} m_j m_k \hat{\mathcal{H}}_\theta(\boldsymbol{x_j}, \boldsymbol{x_k}). \tag{34}$$

We embed (34) into the symplectic integrator that includes $m_j$ to obtain the final network architecture.

The setup of N-body problem is similar to the previous problems. The training period is $T_{train} = 0.08$ and the prediction period is $T_{predict} = 2\pi$. Similar to the setup of previous problems, the learning rate is decaying every 10 epochs. Learning rate, $\gamma$, $i$, $step\_size$, and $M$ are the same as the setup of Kepler problem in Table 1, except we use 40 samples to train our model. The training process takes about 100 epochs for the loss to converge. In figure 12, we use our trained model to predict the dynamics of a 3-body system and a 6-body system. In both cases, our model can predict the paths accurately, with the predicted paths in the 3-body system matching the true paths perfectly. The success of these tasks shows the strong generalization ability of our model. Based on our experiments, our model can be applied to problems with a larger scale, for example, to predict the motions of hundreds of bodies.

## 5 Conclusion

We present Taylor-nets, a novel neural network architecture that can conduct continuous, long-term predictions based on sparse, short-term observations. Taylor-nets consist of two sub-networks, whose outputs are combined using a fourth-order symplectic. Both sub-networks are embedded with the form of Taylor series expansion where each term is designed as a symmetric structure. Our model is able to learn the continuous-time evolution of the target systems while simultaneously preserving their symplectic structures. We demonstrate the efficacy of our Taylor-net in predicting a broad spectrum of Hamiltonian dynamic systems, including the pendulum, the Lotka–Volterra, the Kepler, and the Hénon–Heiles systems.

We evaluate the performance of using the Taylor series as the underlying structure of Taylor-net by comparing it with the most used activation function, ReLU. The experimental results show that the neural networks perform better with

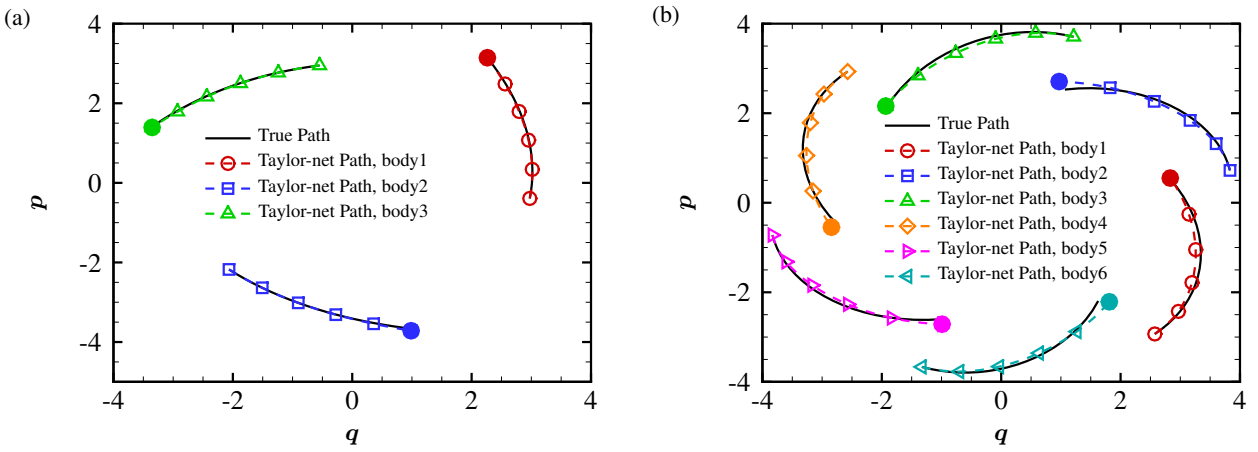


**Figure 12:** Predicted position $\boldsymbol{q}$ and momentum $\boldsymbol{p}$ from $t = 0$ to $t = 2\pi$ (a) for 3 bodies and (b) for 6 bodies. In both (a) and (b), the training period is $T_{train} = 0.08$, and the prediction period is $T_{predict} = 2\pi$. We use the same trained model to make the predictions in (a) and (b), which is trained for 100 epochs.

Taylor series than with ReLU in the pendulum, the Lotka–Volterra, and the Kepler problems. In all three systems, the training loss of using the Taylor series is 10 to 100 times smaller than that of using ReLU. The strong representation ability of the Taylor series is an important factor that increases the accuracy of the prediction.

Moreover, we compare Taylor-net with other state-of-art methods, ODE-net and HNN, to access its predictive ability and robustness. We observe that the prediction error of Taylor-net over the prediction period is half of that of HNN and one-seventh of that of ODE-net. The predictions made by HNN and ODE-net also diverge from the true flow much faster as time increases. Additionally, to test the robustness of our model, we implement two testing cases with small and large noises. We add noise $\sigma_1, \sigma_2 \sim \mathcal{N}(0, 0.1)$ in the case of small noise and $\sigma_1, \sigma_2 \sim \mathcal{N}(0, 0.5)$ in the case of large noise. In the first case, Taylor-net discovers the unknown trajectory successfully, while ODE-net diverges away from the true value quickly. Although the predicting result of HNN does not seem to drift away from the true dynamics, it does not fit the true trajectory as well as the prediction made by Taylor-net. The prediction error of Taylor-net is about two-thirds and half of that of HNN and ODE-net respectively. The difference becomes clearer as we increase the noise. We observe that Taylor-net still makes predictions that are almost consistent with the true trajectories, while ODE-net completely fails to do so. Moreover, the prediction made by HNN is much worse than in the case of small noise, while the performance of Taylor-net remains as good as the previous case. The prediction error of Taylor-net is about half and one-twentieth of that of HNN and ODE-net respectively.

Additionally, we highlight the small training sample size and the fast convergence rate of our model. Under the same setting, HNN and OED-net need 5 times more samples than our model does to achieve the same validation loss, and their models take 10 times and 70 times more epochs to converge. We also test our model under only 1 epoch of training, the prediction results made by HNN and ODE-net completely fail to match the true flow, while Taylor-net predicts the truth to a level that is incomparable with HNN and ODE-net. Compared with HNN and OED-net, our model exhibits its unique computational merits by using small data with a short training period (6000 times shorter than the predicting period), small sample sizes, and no intermediate data to train the networks while outperforming others regrading the prediction accuracy, convergence rate, and robustness to a great extent.

Towards the end of our work in section 4, we discussed the N-body system, which is a high-dimensional Hamiltonian system whose underlying governing equations are non-differentiable. In our future works, we will continue to explore solving this kind of high-dimensional problems, using some essential ideas of Taylor-nets with potential modifications. An other interesting direction will be to design a different neural network architecture with the same structure-preserving ability to learn the dynamics of non-separable Hamiltonian systems.

## Acknowledgments

This project is support in part by Neukom Institute CompX Faculty Grant, Burke Research Initiation Award, and NSF MRI 1919647. Yunjin Tong is supported by the Dartmouth Women in Science Project (WISP), Undergraduate Advising and Research Program (UGAR), and Neukom Scholars Program. Our code is available at `https://github.com/ytong6/Taylor-net`.

## References


[1] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, pages 4502–4510, 2016.

[2] S. L. Brunton, B. R. Noack, and P. Koumoutsakos. Machine Learning for Fluid Mechanics. *Annu. Rev. Fluid Mech.*, 52:477–508, 2020.

[3] Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proc. Natl. Acad. Sci. U.S.A.*, 113(15):3932–3937, 2016.

[4] J. Candy and W. Rozmus. A symplectic integration algorithm for separable Hamiltonian functions. *J. Comput. Phys.*, 92:230–256, 1991.

[5] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud. Neural ordinary differential equations. In *Conference on Neural Information Processing Systems*, pages 6571–6583, 2018.

[6] Z. Chen, J. Zhang, M. Arjovsky, and L. Bottou. Symplectic recurrent neural networks. In *International Conference on Learning Representations*, 2020.

[7] Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian neural networks. *arXiv:2003.04630*, 2020.

[8] D. M. DiPietro, S. Xiong, and B. Zhu. Sparse symplectically integrated neural networks. In *Conference on Neural Information Processing Systems*, 2020.

[9] Yuwei Fan, Cindy Orozco Bohorquez, and Lexing Ying. Bcr-net: A neural network based on the nonstandard wavelet form. *J. Comput. Phys.*, 384:1–15, 2019.

[10] Jordi Feliu-Faba, Yuwei Fan, and Lexing Ying. Meta-learning pseudo-differential operators with deep neural networks. *J. Comput. Phys.*, 408:109309, 2020.

[11] K. Feng and M. Qin. *Symplectic Geometric Algorithms for Hamiltonian Systems*. Springer, 2010.

[12] E. Forest and R. D. Ruth. Fourth-order symplectic integration. *Physica D*, 43:105–117, 1990.

[13] Zhenglin Geng, Daniel Johnson, and Ronald Fedkiw. Coercing machine learning to output physically accurate results. *J. Comput. Phys.*, 406:109099, 2020.

[14] S. Greydanus, M. Dzamba, and J. Yosinski. Hamiltonian neural networks. In *Conference on Neural Information Processing Systems*, pages 15379–15389, 2019.

[15] Constantin Grigo and Phaedon-Stelios Koutsourelakis. A physics-aware, probabilistic machine learning framework for coarse-graining high-dimensional systems in the small data regime. *J. Comput. Phys.*, 397:108842, 2019.

[16] M. Gulian, M. Raissi, P. Perdikaris, and G. Karniadakis. Machine learning of space-fractional differential equations. *SIAM J. Sci. Comput.*, 41(4):A248–A2509, 2018.

[17] E. Hairer, S.P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I – Nonstiff Problems*. Springer, 1987.

[18] W. R. Hamilton. On a general method in dynamics. *Philos. T. Roy. Soc.*, 124:247–308, 1834.

[19] L. N. Hand and J. D. Finch. *Analytical Mechanics*. Cambridge University Press, 2008.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. pages 770–778, 2016.

[21] Quercus Hernandez, Alberto Badias, David Gonzalez, Francisco Chinesta, and Elias Cueto. Structure-preserving neural networks. *arXiv:2004.04653*, 2020.

[22] Alexander Holiday, Mahdi Kooshkbaghi, Juan M Bello-Rivas, C William Gear, Antonios Zagaris, and Ioannis G Kevrekidis. Manifold learning for parameter reduction. *J. Comput. Phys.*, 392:419–431, 2019.

[23] T. W. Hughes, I. A. D. Williamson, M. Minkov, and S. Fan. Wave physics as an analog recurrent neural network. *Sci. Adv.*, 5:6946, 2019.

[24] P. Jin, A. Zhu, G. E. Karniadakis, and Y. Tang. Symplectic networks: intrinsic structure-preserving networks for identifying Hamiltonian systems. *arXiv:2001.03750*, 2020.

[25] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[26] W. Kutta. Beitrag zur näherungsweisen integration totaler differentialgleichungen. *Z. Math. Phys.*, 46:435–453, 1901.

[27] Y. Li, Y. He, J. Niesen Y. Sun, H. Qin, and J. Liu. Solving the vlasov–maxwell equations using hamiltonian splitting. *J. Comput. Phys.*, 396:381–399, 2019.

[28] Julia Ling, Reese Jones, and Jeremy Templeton. Machine learning strategies for systems with invariance properties. *J. Comput. Phys.*, 318:22–35, 2016.

[29] A. T. Mohan, N. Lubbers, D. Livescu, and M. Chertkov. Embedding hard physical constraints in convolutional neural networks for 3d turbulence. In *International Conference on Learning Representations*, 2020.

[30] P. J. Morrison. Hamiltonian and action principle formulations of plasma physics. *Phys. Plasmas*, 12:058102, 2005.

[31] G. Pang, L. Yang, and G. E. Karniadakis. Neural-net-induced Gaussian process regression for function approximation and PDE solution. *J. Comput. Phys.*, 384:270–288, 2019.

[32] M. Raissi and G. E. Karniadakis. Hidden physics models: Machine learning of nonlinear partial differential equations. *J. Comput. Phys.*, 357:125–141, 2018.

[33] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Inferring solutions of differential equations using noisy multi-fidelity data. *J. Comput. Phys.*, 335:736–746, 2017.

[34] 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. *J. Comput. Phys.*, 378:686–707, 2019.

[35] F Regazzoni, L Dedé, and A Quarteroni. Machine learning for fast and reliable solution of time-dependent differential equations. *J. Comput. Phys.*, 397:108852, 2019.

[36] Samuel H Rudy, Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Data-driven discovery of partial differential equations. *Sci. Adv.*, 3(4):e1602614, 2017.

[37] C. Runge. Ueber die numerische auflösung von differentialgleichungen. *Math. Ann.*, 46:167–178, 1895.

[38] D. G. Saari and Z. Xia. *Hamiltonian Dynamics and Celestial Mechanics*. American Mathematical Society, 1 edition, 1996.

[39] R. Salmon. Hamiltonian fluid mechanics. *Ann. Rev. Fluid Mech.*, 20:225–256, 1988.

[40] Alvaro Sanchez-Gonzalez, Victor Bapst, Kyle Cranmer, and Peter Battaglia. Hamiltonian graph networks with ode integrators. *arXiv preprint arXiv:1909.12790*, 2019.

[41] J. M. Sellier, G. M. Caron, and J. Leygonie. Signed particles and neural networks, towards efficient simulations of quantum systems. *J. Comput. Phys.*, 387:154–162, 2019.

[42] J. Sirignano and K. Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *J. Comput. Phys.*, 375:686–707, 2018.

[43] Panos Stinis, Tobias Hagge, Alexandre M Tartakovsky, and Enoch Yeung. Enforcing constraints for interpolation and extrapolation in generative adversarial networks. *J. Comput. Phys.*, 397:108844, 2019.

[44] G. H. Teicherta, A. R. Natarajanc, A. Van der Venc, and K. Garikipati. Machine learning materials physics: Integrable deep neural networks enable scale bridging by learning free energy functions. *Comput. Methods Appl. Mech. Engrg.*, 353:201–216, 2019.

[45] P. Toth, D. J. Rezende, A. Jaegle, S. Racaniére, A. Botev, and I. Higgins. Hamiltonian generative networks. In *International Conference on Learning Representations*, 2020.

[46] V. S. Viswanath and G. Müller. *The Recursion Method*, volume 23. Springer-Verlag, 1994.

[47] S. Xiong, X. He, Y. Tong, R. Liu, and B. Zhu. Roenets: Predicting discontinuity of hyperbolic systems from continuous data. *arXiv:2006.04180*, 2020.

[48] S. Xiong, X. He, Y. Tong, and B. Zhu. Neural vortex method: from finite lagrangian particles to infinite dimensional eulerian dynamics. *arXiv:2006.04178*, 2020.

[49] S. Xiong, Y. Tong, X. He, C. Yang, S. Yang, and B. Zhu. Nonseparable symplectic neural networks. *arXiv:2010.12636*, 2020.

[50] H. Yoshida. Construction of higher order symplectic integrators. *Phys. Lett. A*, 150:262–268, 1990.

[51] Y. D. Zhong, B. Dey, and A. Chakraborty. Symplectic ode-net: learning hamiltonian dynamics with control. In *International Conference on Learning Representations*, 2020.

[52] Aiqing Zhu, Pengzhan Jin, and Yifa Tang. Deep hamiltonian networks based on symplectic integrators, 2020.

[53] Aiqing Zhu, Pengzhan Jin, and Yifa Tang. Inverse modified differential equations for discovery of dynamics, 2020.

# A Adjoint Method

Apply the chain rule to the gradients of loss function and consider the two neural networks $\boldsymbol{T}_p(\boldsymbol{p}, \boldsymbol{\theta}_p)$ and $\boldsymbol{V}_q(\boldsymbol{q}, \boldsymbol{\theta}_q)$ under the framework of neural ODEs, we obtain the following sets of equations:

$$\begin{cases} \dfrac{\partial L}{\partial \boldsymbol{\theta}_p} = \dfrac{\partial L}{\partial \boldsymbol{q}\left(t_1\right)} \dfrac{\mathrm{d}\boldsymbol{q}\left(t_1\right)}{\mathrm{d}\boldsymbol{\theta}_p} \\ \boldsymbol{q}\left(t_1\right) = \displaystyle\int_{t_0}^{t_1} \boldsymbol{T}_p(\boldsymbol{p}, \boldsymbol{\theta}_p)\mathrm{d}t + \boldsymbol{q}_0 \end{cases} \tag{35}$$

$$\begin{cases} \dfrac{\partial L}{\partial \boldsymbol{\theta}_q} = \dfrac{\partial L}{\partial \boldsymbol{p}\left(t_1\right)} \dfrac{\mathrm{d}\boldsymbol{p}\left(t_1\right)}{\mathrm{d}\boldsymbol{\theta}_q} \\ \boldsymbol{p}\left(t_1\right) = -\displaystyle\int_{t_0}^{t_1} \boldsymbol{V}_q(\boldsymbol{q}, \boldsymbol{\theta}_q)\mathrm{d}t + \boldsymbol{p}_0 \end{cases} \tag{36}$$

where $L$ is the loss function, and $\boldsymbol{q}\left(t_1\right)$, $\boldsymbol{p}\left(t_1\right)$, $\boldsymbol{q}_0$, and $\boldsymbol{p}_0$ are $\boldsymbol{q}$ and $\boldsymbol{p}$ at $t_1$ and $t_0$, respectively.

Let $\boldsymbol{b}_p(t) = \mathrm{d}\boldsymbol{q}\left(t\right)/\mathrm{d}\boldsymbol{\theta}_p$ and $\boldsymbol{b}_q(t) = \mathrm{d}\boldsymbol{p}\left(t\right)/\mathrm{d}\boldsymbol{\theta}_q$, we derive the following equations:

$$\begin{aligned} \boldsymbol{b}_p(t) = \frac{\mathrm{d}\boldsymbol{q}\left(t\right)}{\mathrm{d}\boldsymbol{\theta}_p} &= \int_{t_0}^{t} \left[\frac{\partial \boldsymbol{T}_p}{\partial \boldsymbol{\theta}_p} + \frac{\partial \boldsymbol{T}_p}{\partial \boldsymbol{p}}\boldsymbol{b}_p(\tau)\right]\mathrm{d}\tau \\ \Longrightarrow \quad &\begin{cases} \dfrac{\mathrm{d}\boldsymbol{b}_p(t)}{\mathrm{d}t} = \dfrac{\partial \boldsymbol{T}_p}{\partial \boldsymbol{\theta}_p} + \dfrac{\partial \boldsymbol{T}_p}{\partial \boldsymbol{p}}\boldsymbol{b}_p(t) \\ \boldsymbol{b}_p(0) = 0, \end{cases} \end{aligned} \tag{37}$$

$$\begin{aligned} \boldsymbol{b}_q(t) = \frac{\mathrm{d}\boldsymbol{p}\left(t\right)}{\mathrm{d}\boldsymbol{\theta}_q} &= -\int_{t_0}^{t} \left[\frac{\partial \boldsymbol{V}_q}{\partial \boldsymbol{\theta}_q} + \frac{\partial \boldsymbol{V}_q}{\partial \boldsymbol{q}}\boldsymbol{b}_q(\tau)\right]\mathrm{d}\tau \\ \Longrightarrow \quad &\begin{cases} \dfrac{\mathrm{d}\boldsymbol{b}_q(t)}{\mathrm{d}t} = -\dfrac{\partial \boldsymbol{V}_q}{\partial \boldsymbol{\theta}_q} - \dfrac{\partial \boldsymbol{V}_q}{\partial \boldsymbol{q}}\boldsymbol{b}_q(t) \\ \boldsymbol{b}_q(0) = 0. \end{cases} \end{aligned} \tag{38}$$

Given $\boldsymbol{b}_p$ and $\boldsymbol{b}_q$, we can rewrite the gradients of loss function as

$$\frac{\partial L}{\partial \boldsymbol{\theta}_p} = \frac{\partial L}{\partial \boldsymbol{q}\left(t_1\right)}\boldsymbol{b}_p(t), \tag{39}$$

and

$$\frac{\partial L}{\partial \boldsymbol{\theta}_q} = \frac{\partial L}{\partial \boldsymbol{p}\left(t_1\right)}\boldsymbol{b}_q(t). \tag{40}$$

However, the scale for solving differential equations of $\boldsymbol{b}_q$ and $\boldsymbol{b}_p$ is too large. We therefore rewrite $\boldsymbol{b}_p$ and $\boldsymbol{b}_q$ as

$$\boldsymbol{b}_p(t) = \boldsymbol{P}_p(t)\int_{t_0}^{t} \boldsymbol{P}_p(\tau)^{-1}\frac{\partial \boldsymbol{T}_p}{\partial \boldsymbol{\theta}_p}\mathrm{d}\tau, \tag{41}$$

and

$$\boldsymbol{b}_q(t) = -\boldsymbol{P}_q(t)\int_{t_0}^{t} \boldsymbol{P}_q(\tau)^{-1}\frac{\partial \boldsymbol{V}_q}{\partial \boldsymbol{\theta}_q}\mathrm{d}\tau. \tag{42}$$

Substitute $\boldsymbol{b}_p$ and $\boldsymbol{b}_q$ into (37), (38), (39), and (40), we obtain two sets of equations:

$$\begin{cases} \dfrac{\partial L}{\partial \boldsymbol{\theta}_p} = \dfrac{\partial L}{\partial \boldsymbol{q}(t_1)}\boldsymbol{P}_p(t_1)\displaystyle\int_{t_0}^{t_1} \boldsymbol{P}_p(t)^{-1}\frac{\partial \boldsymbol{T}_p}{\partial \boldsymbol{\theta}_p}\mathrm{d}t, \\ \dfrac{\mathrm{d}\boldsymbol{P}_p(t)}{\mathrm{d}t} = \dfrac{\partial \boldsymbol{T}_p}{\partial \boldsymbol{p}}\boldsymbol{P}_p(t), \end{cases} \tag{43}$$

and

$$\begin{cases} \dfrac{\partial L}{\partial \boldsymbol{\theta}_q} = -\dfrac{\partial L}{\partial \boldsymbol{p}(t_1)} \boldsymbol{P}_q(t_1) \displaystyle\int_{t_0}^{t_1} \boldsymbol{P}_q(t)^{-1} \frac{\partial \boldsymbol{V}_q}{\partial \boldsymbol{\theta}_q} \mathrm{d}t, \\ \dfrac{\mathrm{d}\boldsymbol{P}_q(t)}{\mathrm{d}t} = -\dfrac{\partial \boldsymbol{V}_q}{\partial \boldsymbol{q}} \boldsymbol{P}_q(t). \end{cases} \tag{44}$$

However, the scale for solving $\boldsymbol{P}_p$ and $\boldsymbol{P}_q$ is still too large. We now consider the adjoint states $\boldsymbol{a}_p(t)$ and $\boldsymbol{a}_q(t)$:

$$\boldsymbol{a}_p(t) = \frac{\partial L}{\partial \boldsymbol{q}(t_1)} \boldsymbol{P}_p(t_1) \boldsymbol{P}_p(t)^{-1}, \quad \boldsymbol{a}_p(t_1) = \frac{\partial L}{\partial \boldsymbol{q}(t_1)}, \tag{45}$$

and

$$\boldsymbol{a}_q(t) = \frac{\partial L}{\partial \boldsymbol{p}(t_1)} \boldsymbol{P}_q(t_1) \boldsymbol{P}_q(t)^{-1}, \quad \boldsymbol{a}_q(t_1) = \frac{\partial L}{\partial \boldsymbol{p}(t_1)}. \tag{46}$$

We can then rewrite the gradient of loss function regarding to $\boldsymbol{\theta}_p$ as

$$\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{\theta}_p} = \frac{\partial L}{\partial \boldsymbol{q}(t_1)} \boldsymbol{P}_p(t_1) \int_{t_0}^{t_1} \boldsymbol{P}_p(t)^{-1} \frac{\partial \boldsymbol{T}_p}{\partial \boldsymbol{\theta}_p} \mathrm{d}t = \int_{t_0}^{t_1} \boldsymbol{a}_p(t) \frac{\partial \boldsymbol{T}_p}{\partial \boldsymbol{\theta}_p} \mathrm{d}t. \tag{47}$$

Similarly, the gradient of loss function regarding to $\boldsymbol{\theta}_q$ can be derived with the result differs by the sign

$$\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{\theta}_q} = -\int_{t_0}^{t_1} \boldsymbol{a}_q(t) \frac{\partial \boldsymbol{V}_q}{\partial \boldsymbol{\theta}_q} \mathrm{d}t. \tag{48}$$

We now want to derive the derivative of $\boldsymbol{a}_p(t)$ and $\boldsymbol{a}_q(t)$. The derivative of $\boldsymbol{a}_p(t)$ can be derived as follows

$$\begin{aligned} \frac{\mathrm{d}\boldsymbol{a}_p}{\mathrm{d}t} &= \frac{\partial L}{\partial \boldsymbol{q}(t_1)} \boldsymbol{P}_p(t_1) \frac{\mathrm{d}\boldsymbol{P}_p(t)^{-1}}{\mathrm{d}t} \\ &= -\frac{\partial L}{\partial \boldsymbol{q}\left(t_1\right)} \boldsymbol{P}_p\left(t_1\right) \boldsymbol{P}_p(t)^{-1} \frac{\mathrm{d}\boldsymbol{P}_p(t)}{\mathrm{d}t} \boldsymbol{P}_p(t)^{-1} \\ &= -\boldsymbol{a}_p(t) \frac{\mathrm{d}\boldsymbol{P}_p(t)}{\mathrm{d}t} \boldsymbol{P}_p(t)^{-1} \\ &= -\boldsymbol{a}_p(t) \frac{\partial \boldsymbol{T}_p}{\partial \boldsymbol{p}}. \end{aligned} \tag{49}$$

The derivative of $\boldsymbol{a}_q(t)$ can be found in a similar manner. We obtain that

$$\frac{\mathrm{d}\boldsymbol{a}_q}{\mathrm{d}t} = \boldsymbol{a}_q(t) \frac{\partial \boldsymbol{V}_q}{\partial \boldsymbol{q}}. \tag{50}$$

Combine the results we found in (45), (46), (47), (48), (49), and (50), we obtain the sets of equations that are our final result

$$\begin{cases} \dfrac{\partial L}{\partial \boldsymbol{\theta}_p} = \displaystyle\int_{t_0}^{t_1} \boldsymbol{a}_p(t) \frac{\partial \boldsymbol{T}_p}{\partial \boldsymbol{\theta}_p} \mathrm{d}t, \\ \dfrac{\mathrm{d}\boldsymbol{a}_p}{\mathrm{d}t} = -\boldsymbol{a}_p(t) \dfrac{\partial \boldsymbol{T}_p}{\partial \boldsymbol{p}}, \\ \boldsymbol{a}_p(t_1) = \dfrac{\partial L}{\partial \boldsymbol{q}(t_1)}, \end{cases} \tag{51}$$

$$\begin{cases} \dfrac{\partial L}{\partial \boldsymbol{\theta}_q} = -\displaystyle\int_{t_0}^{t_1} \boldsymbol{a}_q(t) \frac{\partial \boldsymbol{V}_q}{\partial \boldsymbol{\theta}_q} \mathrm{d}t, \\ \dfrac{\mathrm{d}\boldsymbol{a}_q}{\mathrm{d}t} = \boldsymbol{a}_q(t) \dfrac{\partial \boldsymbol{V}_q}{\partial q}, \\ \boldsymbol{a}_q(t_1) = \dfrac{\partial L}{\partial \boldsymbol{p}(t_1)}. \end{cases} \tag{52}$$

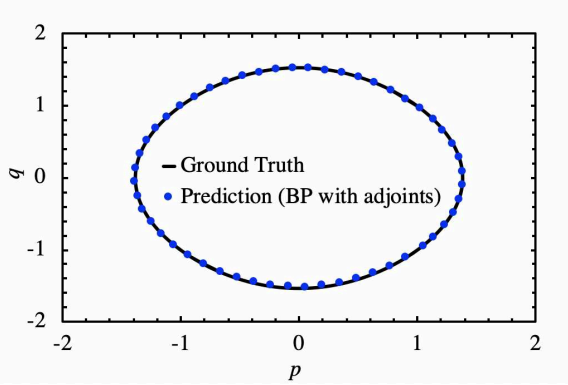


**Figure 13:** Prediction result of the pendulum problem using adjoint method as backward propagation

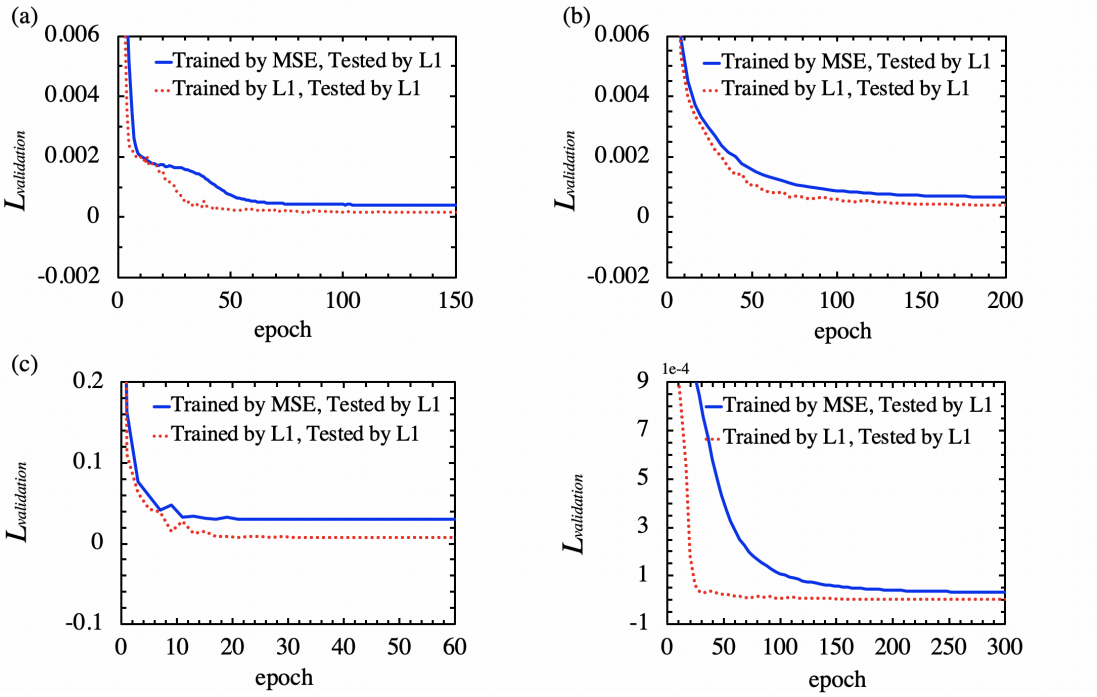


**Figure 14:** Comparisons of validation losses with different training loss functions for (a) the pendulum, (b) the Lotka–Volterra, (c) the Kepler, and (d) the Hénon–Heiles problems validated by L1 loss function. The red dashed lines represent the networks trained by L1 loss function; the blue solid lines represent the networks trained by MSE loss function.

Using (51) and (52), we calculate the gradients of loss function in the backward propagation.

Figure 13, shows the prediction result of the pendulum problem using the adjoint method as our backward propagation method. We can see that the prediction result matches the ground truth well. However, training using the adjoint sensitivity method is about 30 percent slower than training using the automatic differentiation method due to higher time complexity.

## B Loss function ablation test

We conduct the ablation test on the pendulum, the Lotka–Volterra, the Kepler, and the Hénon–Heiles problems to compare the validation loss after convergence with different training loss functions in the training process. Figure 14 shows the comparison of validation losses with different training loss functions in the training process of different

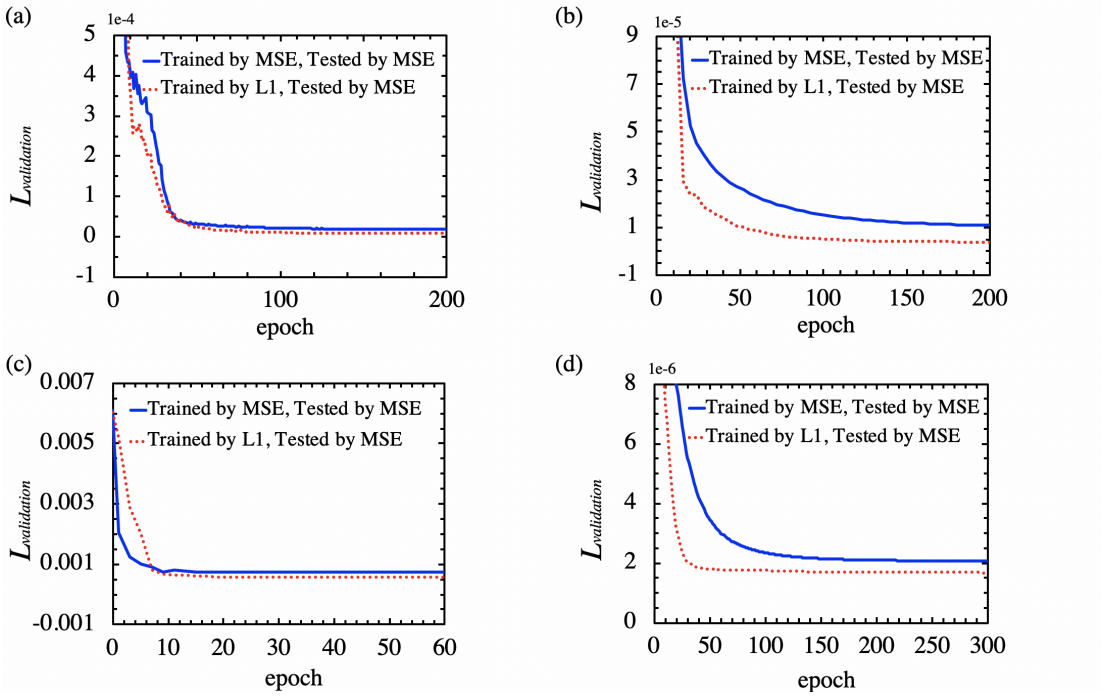


**Figure 15:** Comparisons of validation losses with different training loss functions for (a) the pendulum, (b) the Lotka–Volterra, (c) the Kepler, and (d) the Hénon–Heiles problems validated by MSE loss function. The red dashed lines represent the networks trained by L1 loss function; the blue solid lines represent the networks trained by MSE loss function.

problems validated by L1 loss function. Figure 15 shows the comparison of validation losses with different training loss functions in the training process of different problems validated by MSE loss function. We observe that for all problems, the validation loss with $L1$ is smaller than that with MSE after convergence. The better performance of $L1$ may be due to MSE loss's high sensitivity to outliers. This explains why we choose $L1$ loss function as our training loss function.