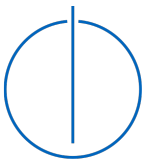
TECHNICAL UNIVERSITY OF MUNICH

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY – INFORMATICS

Master's Thesis in Informatics

Sampling Neural Networks to Approximate Hamiltonian Functions

Atamert Rahma



TECHNICAL UNIVERSITY OF MUNICH

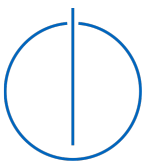
SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY – INFORMATICS

Master's Thesis in Informatics

Sampling Neural Networks to Approximate Hamiltonian Functions

Randomisierte Neuronale Netze zur Approximation von Hamilton-Funktionen

Author:	Atamert Rahma
Supervisor:	Prof. Dr. Felix Dietrich
Advisor:	M.Sc. Chinmay Datar
Submission Date:	15.05.2024



I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15.05.2024 Atamert Rahma

Acknowledgments

I would like to express my deepest gratitude to my supervisor, Prof. Dr. Felix Dietrich, and my advisor, M.Sc. Chinmay Datar, for their guidance and insightful feedback throughout the journey of completing this thesis. Their expertise, constructive criticism, encouragement, and patience have been instrumental in shaping this work.

Also, I gratefully acknowledge the computational and data resources provided by the Leibniz Supercomputing Centre (www.lrz.de) for this Master's thesis.

Abstract

Approximating dynamical systems from data is a significant and challenging problem. Incorporating knowledge about physical laws that govern the dynamical process can help reduce data requirements and improve prediction accuracy. Here, we discuss how to approximate Hamiltonian functions of energy-conserving dynamical systems by solving an associated linear partial differential equation. We employ neural network activation functions as basis functions for the solution and evaluate the performance of data-agnostic and data-driven weight sampling algorithms to construct this basis. We experiment with single pendulum, Lotka-Volterra, double pendulum, and Hénon-Heiles systems and evaluate the approximation capabilities of the sampled networks. The accuracy of the sampled networks using the data-driven sampling scheme outperforms the data-agnostic sampling in various settings where the gradients of the target function get large. Also, the data-agnostic scheme needs proper bias configuration beforehand; otherwise, its accuracy is worse than the data-driven sampling scheme. Finally, we demonstrate how the sampled networks can also learn from discrete trajectory observations using a post-training correction step, which has been studied with traditional neural networks before.

Contents

1 Introduction

Machine learning (ML) [8] techniques have been widely studied in the dynamic landscape of research and technology in the past decade, from healthcare to finance and e-commerce [93]. Amid this ML revolution, neural networks (NNs) [37] have become widely utilized in many application domains, such as robotics [70, 3], reinforcement learning [118], language comprehension [24] and translation [102], image recognition [50] and classification [64], and video game playing [78, 99]. To learn inherent patterns within data and extend the generalization capability beyond the training set, ML techniques commonly integrate a series of inductive biases, such as priors in a Bayesian model [6, 49]. These aim to guide the learning algorithm in learning the approximation that aligns with the established biases. For example, in dimensionality reduction, a priori information about the high-dimensional manifold's global geometry can guide the reduction process and ensure that the selected neighborhood effectively captures the underlying structure [108]. In the case of approximating physical systems, priors can again play an important role in capturing the nature of the system, i.e., its dynamics and physical laws [114, 5, 13]. Physical priors have been integrated into NNs with physics-informed NNs (PINNs) using the loss formulation [86]. Similarly, for Hamiltonian functions of energy-conserving dynamical systems governed by Hamilton's equations [45, 46], learning the dynamics directly from data using NN by integrating physical conservation laws and invariances of the system's nature into the learning algorithm has been recently studied in many work [40, 7, 111, 18, 122, 121, 103, 47, 110, 60, 116, 100] and more. Such data-driven approaches also can discover new physical concepts [57, 96]. We focus on approximating Hamiltonian functions from data by solving an associated linear system built using physical priors. The linear system is based on the one demonstrated for Gaussian Processes (GPs) [88] in [7], and we modify it to utilize NN activation functions instead.

In traditional iterative gradient-descent-based (GD-based) training methods, learning may suffer from large training times [58, 116]. Sampling algorithms can mitigate time-consuming training by sampling the weights and biases and reach orders of magnitude faster training [10]. Here, we employ data-agnostic [56] and data-driven [10] algorithms to sample NNs and use the activation functions as basis functions for the solution. We evaluate both sampling strategies and analyze the advantages of the data-driven sampling scheme in various settings.

The rest of the document is structured as follows: the state-of-the-art in section 2, including the preliminaries (2.1), and review of literature regarding learning Hamiltonian system dynamics (2.2) and NN sampling (2.3); the main body in section 3, which presents a comprehensive framework for learning the Hamiltonian utilizing sampled NNs (3.1), along with numerical experiments on various Hamiltonian systems such as single pendulum, Lotka-Volterra [31], double pendulum, and Hénon-Heiles [51] systems (3.2). Additionally, we explore adapting the framework for learning from discrete trajectory observations (without knowing the true gradients or vector fields) with a post-training correction step (3.3). Finally, the conclusions in section 4 provide a summary and outline future directions for this work.

2 State of the Art

In this section, we cover the state-of-the-art methods for learning Hamiltonian systems using ML methods. We start by introducing the preliminaries in section 2.1 with a short introduction to dynamical systems (2.1.1), state-of-the-art numerical integrators for Hamiltonian systems (2.1.2), least squares approximation (2.1.3) and feed-forward (fully connected) NNs (FFNNs) (2.1.4). Then we dive into the research focusing on learning Hamiltonian systems from data in section 2.2 by focusing on the work related to learning the Hamiltonian function (2.2.1), learning the flow map (2.2.2) and their numerical error analysis and correction (2.2.3). Lastly, we review the literature regarding sampling NNs in section 2.3 with data-agnostic sampling algorithms (2.3.1) and data-driven sampling algorithms (2.3.2).

2.1 Preliminaries

2.1.1 Dynamical Systems

General Definition

A dynamical system is a mathematical representation of the broader scientific concept of a deterministic process [28]. It involves formalizing the prediction of future and past states for various systems from disciplines such as physics, chemistry, biology, ecology, and economy based on their current state and the governing laws of their evolution. Assuming these laws remain constant over time, the system's behavior becomes entirely determined by its initial state. In essence, a dynamical system encompasses a set of potential states (state space) and a law dictating how the system evolves. The evolution of a dynamical system refers to a transformation in the system's state over time, denoted by $t \in T$, where T represents a set of numbers. There are two types of dynamical systems regarding time,

- Continuous time dynamical systems: those operating with continuous (real) time $T = \mathbb{R}$.
- Discrete time dynamical systems: those operating with discrete (integer) time $T = \mathbb{Z}$.

The fundamental element of a dynamical system is a governing law that determines the system's state x_t time t, given the initial state x_0. The general approach to defining this evolution assumes a map φ^t is defined in the state space $\mathcal{X}$ as

$$\varphi^t : \mathcal{X} \to \mathcal{X},$$

which transforms an initial state $x_0 \in \mathcal{X}$ into some state $x_t \in \mathcal{X}$ at time t, i.e.,

$$x_t = \varphi^t(x_0).$$

Frequently referred to as the evolution operator, φ^t may be explicitly known, but often, it is indirectly defined and can only be approximated. In the context of continuous time systems, the family of evolution operators $\{\varphi^t\}_{t\in T}$ forms what is termed a flow.

The evolution operators possess two inherent characteristics that capture the deterministic nature of dynamical systems. Firstly, there is the property

$$\varphi^0 = id,$$

where id represents the identity map on X, denoted by $id(x) = x$ for all $x \in \mathcal{X}$. The first property implies that the system does not undergo spontaneous changes in its state. The second property of the evolution operator is expressed as

$$\varphi^{t+s} = \varphi^t \odot \varphi^s,$$

which means that

$$\varphi^{t+s}(x) = \varphi^t(\varphi^s(x))$$

for all $x \in \mathcal{X}$ and $t, s \in T$, provided both sides of the equation are defined. In essence, the second property asserts that the outcome of the system's evolution over $t + s$ units of time, starting from a state $x \in \mathcal{X}$, coincides with the result obtained by first allowing the system to change from the state x over s units of time and then evolving it over the subsequent t units of time from the resulting state $\varphi^s x$. This property signifies that the governing law of the system remains constant over time, indicating that the system is autonomous.

In this thesis, we use the symbol φ for the flow map of a continuous time system: $\varphi : \mathcal{X} \to \mathcal{X}$. When discretized, the flow map is usually defined with a time step $\Delta t = h > 0$, for which we then write φ_h.

Hamiltonian Systems

Hamiltonian systems, pioneered by Sir William Rowan Hamilton in the 19th century [45, 46], represent a mathematical framework to define dynamical systems. Originally developed to offer a more unified and comprehensive approach to classical mechanics, Hamiltonian mechanics has found widespread application across diverse branches of physics, ranging from thermodynamics to quantum field theory [89, 91, 107]. Its formulation is based on an energy-like quantity called the "Hamiltonian", which provides an alternative description of a dynamical system's behavior.

First, we look at the relation between dynamical systems and differential equations. Continuous time dynamical systems can be defined by differential equations with the law of evolution of the system defined in terms of velocities $\dot{x}_i = (dx_i(t)/dt)$ as functions of the coordinates in an Euclidean state space $(x_1, x_2, \dots x_d) \in \mathcal{X} = \mathbb{R}^d$,

$$\dot{x}_i = v_i(x_1, x_2, \dots, x_d),\ i = 1, 2, \dots, d,$$

or in vector form,

$$\dot{x} = v(x),\ v : \mathcal{X} \to \mathcal{X},$$

where the function on the right side of the equation is a vector field representing a system of first-order ordinary differential equations (ODEs).

The Hamiltonian function, often referred to as the Hamiltonian, of a d degrees of freedom (DOF) Hamiltonian system incorporates a total of $2d$ ODEs (Eq. (2.1) and Eq. (2.2)) that dictates how the dynamical system evolves. A Hamiltonian system is usually defined on a state space $\mathcal{X} \subseteq E \times E$, also called the phase space, with continuous Euclidean space $E = \mathbb{R}^d$. The corresponding Hamiltonian is then defined as $\mathcal{H} : \mathcal{X} \to \mathbb{R}$. The laws of motion defined in Hamilton's equations incorporate both "position" (q) and "momentum" (p) coordinates as

$$\dot{q} = \frac{\partial \mathcal{H}}{\partial p} = \nabla_p \mathcal{H}, \tag{2.1}$$

$$\dot{p} = -\frac{\partial \mathcal{H}}{\partial q} = -\nabla_q \mathcal{H}, \tag{2.2}$$

where we write $\nabla_q \mathcal{H}$ for $\nabla_q \mathcal{H}(x)$, analogously $\nabla_p \mathcal{H}$ for $\nabla_p \mathcal{H}(x)$ given $x = (q, p) \in \mathcal{X}$, where q, p, $\dot{q}$, $\dot{p}$ are in vector form for simplicity. Additionally, the time subscript (t) is dropped for readability. Hamilton's equations (Eq. (2.1) and Eq. (2.2)) hold for all $x \in \mathcal{X}$ and all $t \in T$.

For this thesis, we shall restrict our attention to autonomous Hamiltonian systems, which do not explicitly depend on time t.

When the Hamiltonian is autonomous, then its value is constant along the trajectories $\{x(t) = (q(t), p(t)) \mid t \in T\}$. We can validate this using Eq. (2.1) and Eq. (2.2),

$$\frac{\mathrm{d}\mathcal{H}}{\mathrm{d}t} = \nabla_q \mathcal{H} \cdot \dot{q} + \nabla_p \mathcal{H} \cdot \dot{p} \overset{\textbf{Eq.}}{=} -\dot{p} \cdot \dot{q} + \dot{q} \cdot \dot{p} = \vec{0} \tag{2.3}$$

for all $x = (q, p) \in \mathcal{X}$, where we again write $\frac{\mathrm{d}\mathcal{H}}{\mathrm{d}t}$ for $\frac{\mathrm{d}\mathcal{H}(x)}{\mathrm{d}t}$ for simplicity, and **Eq.** stands for Eq. (2.1) and Eq. (2.2). This coincides with the energy conservation law defined in many mechanical systems. If the Hamiltonian is additionally separable, one can formulate the Hamiltonian as $\mathcal{H} = \mathcal{T}(p) + \mathcal{U}(q)$, where $\mathcal{T}$ is for kinetic energy, $\mathcal{U}$ for potential energy. In this case, the Eq. (2.1) becomes $\dot{q} = \nabla_p \mathcal{T}$, and Eq. (2.2) becomes $\dot{p} = -\nabla_q \mathcal{U}$.

Eq. (2.1) and Eq. (2.2) can be rewritten to formulate a partial differential equation (PDE) for $\mathcal{H}$ for all $x = (q, p) \in E$, as

$$\mathcal{J} \cdot \nabla \mathcal{H}(x) - v(x) = \vec{0}, \tag{2.4}$$

where $\mathcal{J} = \begin{bmatrix} 0_d & I_d \\ -I_d & 0_d \end{bmatrix} \in \{0,1\}^{2d \times 2d}$, $I_d \in \{0,1\}^{d \times d}$ is an identity matrix, and 0_d is a d by d square matrix where all the entries are equal to zero. Since we can formulate a homogeneous Hamiltonian system as a system of ODEs using Hamilton's equations, general PDEs are out of scope for this thesis.

Note that the Hamiltonian of a dynamical system differs from its evolution operator; however, numerically integrating the equations of motion derived from the Hamiltonian using a numerical integrator allows us to approximate the system's evolution over time. While we are not directly writing the evolution operator φ^t in an analytical form, the numerical integration can effectively approximate the system's trajectory in its phase space. Additionally, knowing the Hamiltonian of a system, we can plot the phase space without explicitly solving the system for a trajectory. We can draw the curves where $\mathcal{H}(x)$ is constant (contour lines), and the solutions of the system should lie on these sets. Since we have the vector field $v(x)$ defined indirectly by Hamilton's Eq. (2.4), we can figure out the directions on the solution sets. Also note that the equilibrium points, where the time derivatives of both q and p are zero, occur at critical points for Hamiltonian systems due to Hamilton's Eq. (2.4) because at critical points the partial derivatives of the Hamiltonian with respect to (w.r.t.) the variables q and p must be zero. Another important property of Hamiltonian systems, as Poincaré pointed out in 1899, is that their flow map is symplectic given $\mathcal{H}$ is twice continuously differentiable on $\mathcal{X} \subset E \times E = \mathbb{R}^d \times \mathbb{R}^d$ (Theorem 2.4 in chapter 6 of [43]). We assume that the condition holds for the Hamiltonians that we focus on in this thesis, i.e., that the flow maps of the Hamiltonians are symplectic (see Def. 2.1.1 for symplectic map).

Definition 2.1.1 (Symplectic Map)**.** A differentiable map function $\varphi : \mathcal{X} \to \mathcal{X}$ ($\mathcal{X} \subseteq E \times E = \mathbb{R}^d \times \mathbb{R}^d$ is an open set) is called **symplectic** if,

$$\varphi'(x)^T \mathcal{J} \varphi(x) = \mathcal{J}$$

for all $x \in \mathcal{X}$, where $\varphi'(x)$ is the Jacobian of $\varphi(x)$ [43].

2.1.2 Numerical Methods for Ordinary Differential Equations

In this section, we provide a brief introduction to the numerical methods for approximating the solution of an ODE ($\dot{x} = v(x)$) from a state x, also known as the initial value problem. The methods approximate the function value $x(t)$ at discrete times $t_0, t_1 \dots t_T$ spaced with $h = \Delta t$. Explicit Euler (also called forward Euler) integrator is stated as

$$x_{n+1} = x_n + h \cdot v(x_n), \tag{2.5}$$

and in the case of Hamiltonian systems, as

$$q_{n+1} = q_n + h \cdot \nabla_p \mathcal{H}(q_n, p_n)$$

for q, and

$$p_{n+1} = p_n - h \cdot \nabla_q \mathcal{H}(q_n, p_n)$$

for p [43].

Symplectic integrators [22, 30, 106] are the proper numerical method for solving the Hamiltonian systems because they are volume-preserving. They are studied in both theoretical [42, 67, 123] and practical [34, 122, 18, 103, 116] studies. The symplectic Euler method for a Hamiltonian system is

$$\begin{aligned} q_{n+1} &= q_n + h \cdot \nabla_p \mathcal{H}(q_{n+1}, p_n), \\ p_{n+1} &= p_n - h \cdot \nabla_q \mathcal{H}(q_{n+1}, p_n), \end{aligned} \tag{2.6}$$

or if implicit for p, then

$$\begin{aligned} p_{n+1} &= p_n - h \cdot \nabla_q \mathcal{H}(q_n, p_{n+1}), \\ q_{n+1} &= q_n + h \cdot \nabla_p \mathcal{H}(q_n, p_{n+1}), \end{aligned}$$

which are semi-implicit and symplectic methods of order 1, as stated in chapter 6 of [43]. We use the first alternative, where q is implicit in this thesis. If the Hamiltonian is separable, then both symplectic Euler methods become explicit in their implementation,

where they can be computed without solving any implicit equation or using fixed point iteration. The first alternative given in Eq. (2.6) can then be written as

$$q_{n+1} = q_n + h \cdot \nabla_p \mathcal{H}(q_{n+1}, p_n) = q_n + h \cdot \nabla_p \mathcal{T}(p_n),$$
$$p_{n+1} = p_n - h \cdot \nabla_q \mathcal{H}(q_{n+1}, p_n) = p_n - h \cdot \nabla_q \mathcal{U}(q_{n+1}).$$

Note how the terms $\nabla_p \mathcal{H}(q_{n+1}, p_n)$ and $\nabla_p \mathcal{H}(q_n, p_n)$ are equal in the case of a separable Hamiltonian, as the partial derivative w.r.t. p does not depend on q. The second alternative is formulated analogously.

Other notable symplectic methods are the implicit midpoint rule of order 2 [43], higher-order symplectic Runge-Kutta methods [42], higher-order Tao's augmented explicit symplectic integrator for separable and non-separable Hamiltonians [106], leapfrog (Newton-Störmer–Verlet-leapfrog) integrator [67, 43]. In learning the flow map of Hamiltonians, integrators that are explicit in their implementations play an important role, as often the loss needs to be backpropagated through the integrator (2.2.2). Leapfrog and symplectic Euler are explicit in their implementations only for separable Hamiltonians. Tao's augmented scheme is explicit in its implementation for separable and non-separable Hamiltonians but operates in an augmented space where the number of dimensions is doubled.

The error analysis shows that the symplectic integrators can preserve the system's energy in long-time integrations. Therefore, other classical numerical integration methods, such as explicit Euler and most of the classical Runge-Kutta and multi-step methods, are not preferred for the long-time integration of Hamiltonian systems [42].

2.1.3 Least Squares Approximation

Definition 2.1.2 (Linear Least Squares Solution)**.** Let $A \in \mathbb{R}^{m \times n}$, and $b \in \mathbb{R}^m$, where $m > n$ (an overdetermined system). A solution $x \in \mathbb{R}^n$ to the minimization problem

$$\min_x \|Ax - b\| \qquad \text{or} \qquad \min_x \|Ax - b\|^2$$

is called **linear least squares solution**, where $\|\cdot\|$ denotes a norm (we use the Euclidean norm in this thesis. Therefore, we avoid writing subscripts for simplicity).

Lemma 2.1.1 (Convexity of Linear Least Squares)**.** *Let $A \in \mathbb{R}^{m \times n}$, and $b \in \mathbb{R}^m$, where $m > n$ (an overdetermined system). The optimization problem $\arg\min_x \|Ax - b\|$ is a **convex optimization problem**.*

Proof. Let $f : x \mapsto \|Ax - b\|^2 = \|b - Ax\|^2 = (b - Ax)^T(b - Ax) = b^T b - 2b^T Ax + (Ax)^T Ax = \|b\|^2 - 2b^T Ax + \|Ax\|^2$ be the function we would like to minimize. We

show that the second-order derivative, i.e., the Hessian, is positive semi-definite by first writing the gradient as

$$\nabla f = -2b^T A + 2(Ax)^T A = -2b^T A + 2x^T A^T A,$$

and then the Hessian as

$$\nabla^2 f = 2A^T A.$$

Since for all $x \in \mathbb{R}^n$ it holds that

$$\begin{aligned} x^T(2A^T A)x \geq 0 &\iff 2x^T A^T Ax \geq 0 \iff \\ 2(Ax)^T Ax \geq 0 &\iff 2\|Ax\|^2 \geq 0, \end{aligned}$$

the Hessian is positive semi-definite, f is a convex function. Therefore, the optimization problem $\arg\min_x f(x)$ is a convex optimization problem. □

In this thesis, we have to deal with overdetermined linear systems that usually do not have an exact solution: $b \notin \mathrm{Im}(A)$, and the matrix A is not square. Thus, we resort to methods for solving least squares solutions. The `lstsq` function from `numpy.linalg` [48] and `scipy.linalg` [112] both are designed to solve least squares solution given rectangle matrix A of an overdetermined system using LAPACK routines [2], which are based on decomposition methods (SVD or QR). We use NumPy's `lstsq`, with the NumPy version (v1.26.3) in this thesis, where *gelsd* routines are used in the `lstsq` implementation, which are based on divide-and-conquer SVD [41].

2.1.4 Feed-Forward Neural Networks

In this thesis, we work with FFNNs configured for regression, i.e., approximating the Hamiltonian. Following the notation in [10] we define an FFNN as follows:

Definition 2.1.3 (FFNN)**.** Let $\mathcal{X} \subseteq \mathbb{R}^D$ be an input space, and $\mathcal{Y} \subseteq \mathbb{R}$ the one-dimensional output space, and $\Phi : \mathcal{X} \to \mathcal{Y}$ is an FFNN defined for regression of a one-dimensional function, with L hidden layers. We write

$$\Phi^{(l)}(x) = \begin{cases} x, & \text{for } l = 0 \\ \sigma(W_l \Phi^{(l-1)}(x) - b_l), & \text{for } 0 < l \leq L \\ W_l \Phi^{(l-1)}(x) - b_l, & \text{for } l = L+1 \end{cases}$$

as the output of the lth layer, where

- $\sigma : \mathbb{R} \to \mathbb{R}$ is the activation function, which is applied element-wise to the input,

- $\{W_l, b_l\}_{l=1}^{L+1}$ are the parameters of the network Φ: weights and biases, where $W_l \in \mathbb{R}^{N_l \times N_{l-1}}$ and $b_l \in \mathbb{R}^{N_l}$,
- N_l is the number of neurons in the lth layer, where $N_0 = D$ for the input layer and $N_{L+1} = 1$ for the output layer for approximating the one-dimensional function.

For $L = 1$, we use the term "shallow network", for $L > 1$, we say "deep network". If we drop the superscript and write $\Phi(x)$, this means the output of the network $\Phi^{(L+1)}(x)$ for an input $x \in \mathcal{X}$.

Remark. The FFNN in Def. 2.1.3 is defined for regression (no activation, i.e., identity for the output layer) and Hamiltonian approximation ($N_{L+1} = 1$).

2.2 Learning Hamiltonian Systems

Classical methods for dynamical systems such as the methods for solving ODEs described in 2.1.2, or for solving PDEs [65] (e.g., finite elements, spectral methods, etc.) have been proven to be robust and efficient in various applications [104, 54, 29, 42]. Many PDE solvers, e.g. arbitrary high-order ADER discontinuous Galerkin (DG) [109], have been recently optimized and successfully applied for specific problems such as flow simulations [32], tsunami simulations [87], seismology [27] and more [90]. Classical methods are the most efficient and scalable computational standard for solving certain PDEs. Still, usually, the methods assume a specific structure in the system's equations, e.g., for the finite element method, the inf–sup condition is assumed for the Navier–Stokes equations [11] and to be able to apply the classical methods we need to know the system's equations beforehand. To encode underlying physical laws directly from data, data-driven approaches such as PINNs are introduced for PDEs [86]. PINNs and other machine learning methods for PDEs, like physics-informed GP, have advantages over classical methods in terms of their simplicity in implementation, being able to discover new governing PDEs, learning reduced-order models, improving existing numerical methods, seamless integration of mathematical models, lack of need for mesh generation [80, 12, 9].

To learn the dynamics of Hamiltonian systems from data using NNs, physical priors of the Hamiltonian systems have been integrated into the network's loss formulation based on Hamilton's Eq. (2.4) [40, 7]. Studies have taken different directions to capture the dynamics of Hamiltonian systems; one focuses on learning the Hamiltonian function of the underlying system, and the other on learning the time evolution of the system by learning the flow map.

2.2.1 Learning the Hamiltonian Function

Most related to our work is the modeling of the Hamiltonian function approximation using a GP by Bertalan et al. [7]. Assuming our dataset consists of a collection $X \subseteq \mathcal{X}$ of K points in the phase space $\mathcal{X}$ with gradients $\nabla\mathcal{H}(x_i) = (\nabla_q\mathcal{H}(x_i), \nabla_p\mathcal{H}(x_i))^T = (-\dot{p}_i, \dot{q}_i)^T \in \mathbb{R}^{2d}$ for all $i \in [K]$, the approximate Hamiltonian value for a new point $y \in \mathcal{X}$ is predicted using the conditional expectation of the GP as

$$E[\widehat{\mathcal{H}}(y)|X, \mathcal{H}(X)] = k(y, X)^T k(X, X)^{-1}\mathcal{H}(X), \tag{2.7}$$

where $\mathcal{H}(X) = (\mathcal{H}(x_1), \mathcal{H}(x_2), \ldots, \mathcal{H}(x_K)) \in \mathbb{R}^K$ is the true function values at all points $x \in X$, $k(X, X) \in \mathbb{R}^{K\times K}$ is the covariance matrix with $k(X, X)_{i,j} = k(x_i, x_j)$ where $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a kernel function, and the symbol $k(y, X) \in \mathbb{R}^K$ used for the evaluation of the kernel function with the new point and the all other points in the dataset X. Given a set Y of M new points, the authors have differentiated Eq. (2.7) and formulated

$$\underbrace{\begin{bmatrix} \nabla_x k(x_1, Y)^T k(Y, Y)^{-1} \\ \nabla_x k(x_2, Y)^T k(Y, Y)^{-1} \\ \vdots \\ \nabla_x k(x_N, Y)^T k(Y, Y)^{-1} \\ k(x_0, Y)^T k(Y, Y)^{-1} \end{bmatrix}}_{\in\mathbb{R}^{(2dK+1)\times M}} \cdot \underbrace{\big[\mathcal{H}(Y)\big]}_{\in\mathbb{R}^M} = \underbrace{\begin{bmatrix} \nabla\mathcal{H}(x_1) \\ \nabla\mathcal{H}(x_2) \\ \vdots \\ \nabla\mathcal{H}(x_K) \\ \mathcal{H}(x_0) \end{bmatrix}}_{\in\mathbb{R}^{(2dK+1)}}, \tag{2.8}$$

where the symbol $k(x_i, Y) \in \mathbb{R}^M$ is used for the evaluation of the kernel function at a point $x_i \in X$ with all other points in the new set Y, and the last row $k(x_0, Y)^T k(Y, Y)^{-1}$ is to fix the integration constant assuming that we know the real function value $\mathcal{H}(x_0)$ for a value x_0 in the phase space. The authors compute the approximate function values $\widehat{\mathcal{H}}(Y)$ using the least squares solution of the linear system in Eq. (2.8). Inspired by this linear system, we construct our architecture in section 3.

The authors additionally propose approximation using NNs for time-independent Hamiltonians. They formulate the loss function as a weighted sum: $\mathcal{L} = \sum_{k=1}^{4} c_k f_k^2$, where

$$f_1 = \dot{q} - \widehat{\dot{q}} = \dot{q} - \nabla_p\widehat{\mathcal{H}}, \qquad f_2 = \dot{p} - \widehat{\dot{p}} = \dot{p} + \nabla_q\widehat{\mathcal{H}}, \tag{2.9}$$

$$f_3 = \mathcal{H}(x_0) - \widehat{\mathcal{H}}(x_0), \qquad f_4 = \frac{\mathrm{d}\widehat{\mathcal{H}}}{\mathrm{d}t} = \nabla_q\widehat{\mathcal{H}} \cdot \dot{q} + \nabla_p\widehat{\mathcal{H}} \cdot \dot{p}, \tag{2.10}$$

where we write hat symbol $\widehat{(\cdot)}$ for the terms that are approximated. The terms f_1 and f_2 incorporate the governing equations of the system: Hamilton's Eq. (2.4), the term

f_3 is again to fix the integration constant assuming that we know the true function for a single data point, the term f_4 incorporates the time-independence of the system, as shown in Eq. (2.3). Except for the term f_3 for fixing the integration constant, the loss is operated on the gradients of the network's output w.r.t. the inputs q and p, which can be computed using automatic differentiation [75]. They also build a pipeline by incorporating an autoencoder [95] that maps between the observation space and the phase space to learn the Hamiltonian from nonlinear high-dimensional observations such as images [7].

Hamiltonian NN (HNN) is introduced by Greydanus et al. [40] in an independent work, which uses a very similar loss as in [7] but uses only the first part that incorporates Hamilton's Eq. (2.4), similar to Eq. (2.9), as

$$\begin{aligned} \mathcal{L}_{\mathbf{HNN}} &= \left\|\dot{q} - \widehat{\dot{q}}\right\| + \left\|\dot{p} - \widehat{\dot{p}}\right\| \\ &= \left\|\dot{q} - \nabla_p \widehat{\mathcal{H}}\right\| + \left\|\dot{p} + \nabla_q \widehat{\mathcal{H}}\right\|, \end{aligned} \tag{2.11}$$

where $\|\cdot\|$ denotes the Euclidean (L^2) norm, and $\widehat{\mathcal{H}}$ is the output of the network, i.e. the conserved quantity. HNN differs from [7] in qualitative analysis, as HNN is evaluated on the trajectories created using the network's approximations (Hamiltonian dynamics using network output's gradients w.r.t. inputs). Bertalan et al. [7] is directly evaluated on the network outputs (Hamiltonian function values). That is why the term for fixing the integration constant (f_3 in Eq. (2.10)) is not necessary for HNN, as it only learns to capture the dynamics of the given Hamiltonian system by incorporating the inductive bias, Hamilton's Eq. (2.4), into the objective function of the network as written in Eq. (2.11), and therefore, does not match the real Hamiltonian of the system. Unlike the baseline model the authors use in [40] for comparison (NODE, see the next section 2.2.2), the objective function is operated on the gradient of the network using automatic differentiation rather than the output itself (see Eq. (2.12) for comparison). Fig. 2.1 depicts HNN's architecture.

The authors show that HNN conserves an energy-like quantity, whereas the baseline model does not. Additionally, when predicting the trajectories on the well-trained HNN, the study shows that HNN could better predict the trajectories than the baseline model [40].

Later, many variations of the HNN have been studied. Lagrangian NN uses the Lagrangian formalism instead of the Hamiltonian as the physics prior and, therefore, does not require canonical coordinates (momenta) [20]. Adaptable HNN is introduced in [47], which incorporates an input channel (bifurcation parameter) that makes the trained model adaptable to parameter variations of the Hénon-Heiles system. This is especially useful if the system can undergo a slow drift or sudden changes in some

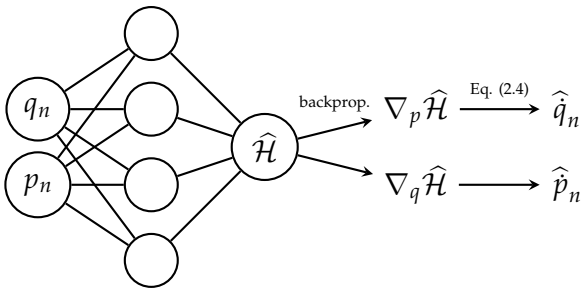


Figure 2.1: HNN defined in [40], outputs the HNN-conserved quantity $\widehat{\mathcal{H}}$.

parameters. Port-HNN additionally incorporates a separate network that depends on time t for controlling force [23]. Dissipative HNN for dissipative Hamiltonian systems includes an additional sub-network that outputs Rayleigh dissipation to account for dissipative effects [100]. Separable HNN incorporates additive separability into the network architecture as an inductive bias [63].

Further studies include learning the coordinate-free symplectic 2-form from data [17], learning a symplectic reduced-order Hamiltonian from high-dimensional data [98, 117].

2.2.2 Learning the Flow

A general framework, neural ordinary differential equations (NODE) by Chen et al. [16], is a framework to learn ODE dynamics on an observed trajectory. NODE approximates the right-hand-side (RHS) of the ODE, $\dot{x} = v(x)$, using an NN and scalably construct the loss through the solver as

$$\mathcal{L}_{\mathbf{NODE}} = \|x_i - \widehat{x}_i\|, \tag{2.12}$$

where $\{x_i\}_{i=0}^T$ denotes the observed trajectory, $\{\widehat{x}_i\}_{i=0}^T$ the predicted trajectory. They use an adjoint ODE to compute the gradients instead of backpropagating through the ODE solver to reduce the memory requirements.

Using time-series data of position q and momentum p only, one can train the HNN by relying on the finite differences, as mentioned in [40]. This would give rise to the forward Euler integration scheme (Eq. (2.5)) in the HNN's objective in Eq. (2.11) and remove the requirement for derivative information in network training. Many recent studies focus on this numerical integration of the HNN, including symplectic recurrent NN (SRNN), introduced by Chen et al. [18]. SRNN outperforms the HNN in predicting long-term trajectories using a higher-order integration scheme. Note that this loss formulation requires backpropagation through the integrator, so an explicit integrator must be used for training. Leapfrog integrator [67, 43] is used in their architecture,

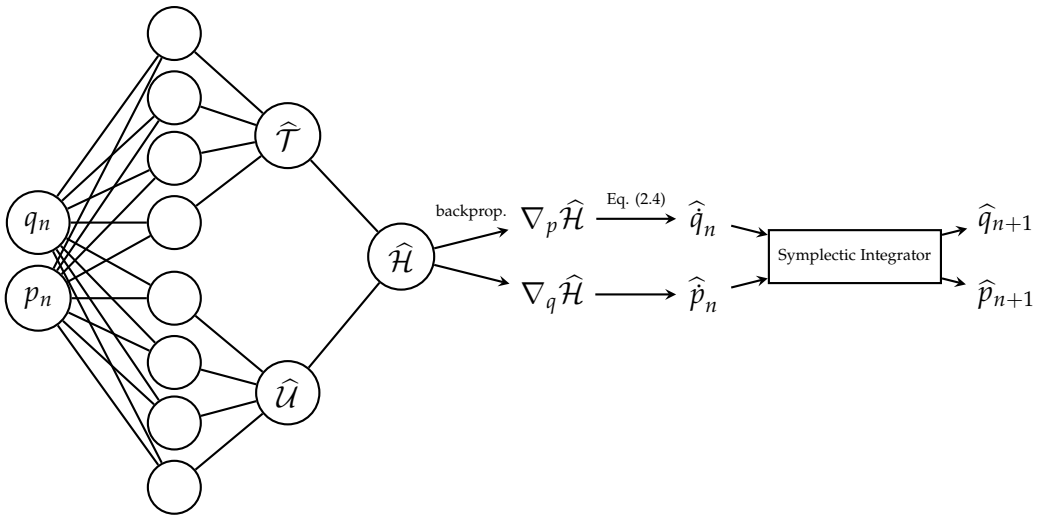


Figure 2.2: SRNN defined in [18].

where the network is split in two, one for kinetic, and one for potential energy as illustrated in Fig. 2.2. Therefore, they only consider separable Hamiltonians. They showed that using the symplectic integrator leapfrog in the network architecture is preferred over using the non-symplectic integrator forward Euler. With the symplectic integrator, the network's predictions outperform trajectories simulated by the exact Hamiltonian equations. This may be because the network learns to minimize the numerical error made by the integrator [18, 52]. SymODEN and dissipative SymODEN also learn with a numerical integrator but can additionally account for external force (e.g., control) [122, 121]. Inspired by SymODEN, capturing SE(3) kinematics for robotic applications are studied in [26]. Stiffness-aware HNN defines a stiffness metric to classify the given trajectories as stiff or non-stiff [71]. Based on this metric, they adapt the step size of the integrator used inside the network. Using the SRNN architecture and the fourth-order symplectic integration schema in [33], DiPietro et al. report good performance on noisy and sparse datasets [25]. Using the extended explicit high-order integrator scheme, Tao's integrator [106], designed for both separable and non-separable Hamiltonians, Xiong et al. [116] use an extended phase space with an augmented Hamiltonian to also handle non-separable Hamiltonians. As a downside, the augmented Hamiltonian has twice as many dimensions as the original Hamiltonian form. Therefore, they must double the phase space dimensions. Deep energy-based modeling DGNet focuses on solving general conservative PDEs from discrete-time data [76]. Graph NNs have been studied in Hamiltonian graph network (HGN) with similar ideas of solving the Hamiltonian equations and passing them through an integrator [92].

Generalization through untrained time steps and higher-order integrators have been studied. It is reported that the HGN could more accurately match the true Hamiltonian using a higher-order integrator. Since the above methods backpropagate through an integrator, their training is reportedly more time-consuming [116].

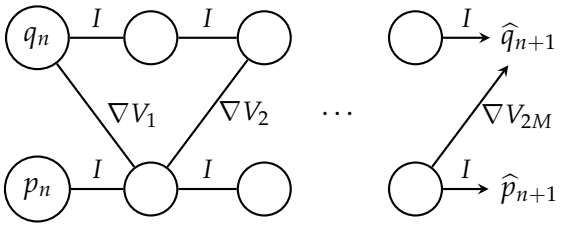


Figure 2.3: Example flow map using SympNet with $2M$ alternating gradient modules ∇V_i, and I being the identity matrix [103].

SympNet is developed to approximate symplectic maps (see Def. 2.1.1) with universal approximation properties, including the non-separable Hamiltonian systems' symplectic flow map from time series data [103]. Unlike previously mentioned studies, SympNet directly focuses on learning the symplectic flow map by concatenating symplectic layers. As the concatenation of symplectic layers is again symplectic, the learned map is again symplectic. The authors note that training is much faster than HNN methods and outperforms them in trajectory prediction. However, due to the number of parameters in SympNet, it is hard to scale it to higher dimensional problems. Horn et al. [52] compare SympNets and HNN architectures and their performance against classical multilayer perceptrons (MLPs). They note that SympNet performs much better outside the training data than MLPs, but they are not more stable inside the range of the training data. The authors also analyze SympNet architecture portrayed in Fig. 2.3 and topological similarities between SympNet and HNN architecture. They note that although SympNet focuses on learning the flow map directly, every two modules of SympNet learn a hidden Hamiltonian inside. Later, motivated by SympNet, TaylorNet is introduced for separable Hamiltonian systems, which can predict sparse and noisy observations and can be extended to higher-dimensional systems [110].

Further studied include, weak form generalized learning for odd-dimensional systems – with an additional contribution to avoid backpropagating through an ODE solver [19], Poisson NN for learning the phase flow of Poisson systems [61], learning using orthogonal polynomials instead of NNs [115], hybrid method for using network approximations for the computationally expensive parts in higher dimensional Hamiltonian systems [94], learning the flow from high-dimensional observations like images [111], HNN embedded GAN-based [38] video generation where each video frame is an observation of the system state and state transitions are determined by

Hamilton's equations [1], equation-driven HNN where the Hamiltonian is assumed to be known [77], Bayesian system identification using symplectic integrators with uncertainty assessments [34]. Random feature models [56, 84] have also been studied for learning Hamiltonian flow in [60] and [105]. However, Tanaka et al. [105] use GP with symplectic random Fourier features integrated with Hamiltonian mechanics, and Jakovac et al. [60] use the extreme learning machine idea, which both are data-agnostic sampling schemes as we explain in later section 2.3.1.

2.2.3 Error Analysis and Correction

Zhu et al. [123] theoretically analyze the training of HNNs without the knowledge of the gradients or vector fields using finite differences with symplectic (symplectic Euler and implicit midpoint rule) and non-symplectic (forward Euler and trapezoidal rule) integration schemes using trajectory data $\mathcal{D} = \{(x_i, \varphi_h(x_i))\}_{i=1}^{K}$ observations only. They show that the error between the conserved quantity predicted by the HNN and the real Hamiltonian depends on the accuracy order of the used integrators. Using symplectic integrators, the predicted trajectories (network targets) have higher accuracy; conversely, the networks trained with symplectic integrator schemes learn the network targets rather than the real Hamiltonian of the original system. The theoretical work in [123] aligns with the numerical results in [18], where the network learns to account for the numerical errors.

Additionally, inverse-modified equations and inverse-modified Hamiltonian are studied with symplectic integrators in [123], such that if the integrator is applied to the inverse-modified Hamiltonian, one gets the true phase flow φ_h of the system. An adapted loss is proposed to learn the inverse-modified Hamiltonian up to an order using its formal series. Similarly, the GP solution in Eq. (2.8) is extended to learn the inverse-modified Hamiltonian only from trajectories without the knowledge of true gradients or vector fields [79]. In an independent study, David et al. confirm that no exact Hamiltonian can be learned from observations by HNN due to integration errors [21]. Similar to Zhu et al. [123], the authors use the formal series expansion of the HNN output trained with symplectic integrators and propose a post-training correction step to the HNN output to recover the original Hamiltonian of the system. We will use the same post-training correction scheme and demonstrate in the main section 3.3 that it can also be applied to sampled networks in our framework.

2.3 Sampling Neural Networks

FFNN architectures for solving classical ML problems such as classification and regression and their universal approximation capabilities have been extensively studied [53,

83, 68, 4]. Yet, traditional iterative training methods for NNs include backpropagation and usually GD [14], which comes with challenges like slow convergence [58], learning rate sensitivity [73] and not finding the global optima [39]. To overcome this, random feature models (RFMs) can be utilized, where the majority of the parameters of the network are either randomized or fixed according to some heuristic [119, 101].

2.3.1 Random Feature Models

Shallow FFNN with random weights (SNN) was proposed in 1992 [97] where the hidden layer weights are sampled randomly from a uniform distribution, and only the output weights and biases are learned. The random vector functional link net additionally incorporates a direct link from the input to the output layer [81]. Extreme learning machine (ELM) is similar to SNN but includes no output biases to be learned [56]. We refer to the RFM method used in the main section 3 as ELM because of its popularity in research. Still, we also learn the output biases, as we explain the exact architecture in the main section. Fixing the hidden layer parameters reduces the non-convex optimization problem into a convex optimization problem, which can be computed via a closed-form solution [82, 56]. However, by allowing only a part of the NN to be trained, the approximation capabilities of RFMs can be questioned. These questions have been extensively addressed, showing the approximation capabilities and error bounds of such random methods [55, 85, 120, 69]. Many variants of ELM have been introduced addressing the problems of increasing learning speed, reducing computation time and model complexity [74], and ELM could be applied to classical ML tasks such as classification, regression, and unsupervised learning (dimensionality reduction, clustering) with good generalization [113]. In small to medium-sized problems, ELM can achieve a significant trade-off between accuracy and training time [35]. Other randomized methods include random features for kernel method [84, 85, 62, 66, 44] and reservoir computing [59, 72].

In the context of dynamical systems, RFMs are studied in learning both low-dimensional [15] and high-dimensional PDEs without the curse of dimensionality [36], and learning Hamiltonian flow [60, 105].

2.3.2 Sample Where It Matters (SWIM)

Aside from their usability in various settings and training speed, RFMs usually come with a catch. To use these methods, one must first choose a high-dimensional probability distribution, and random methods usually rely on a normal distribution, which is independent of the specific problem or data. Regarding accuracy, random methods usually require thousands of neurons to compete with the traditional GD-based meth-

ods and show less performance on large-scale problems [35]. To address these issues, a data-driven sampling algorithm, "Sample Where It Matters (SWIM)," is introduced in [10]. The sampling schema used in SWIM is data-driven (see Def. 2.3.1 for its formal definition, and Fig. 2.4 for intuitive understanding). SWIM networks can also account for deep NNs, making their constructions very efficient.

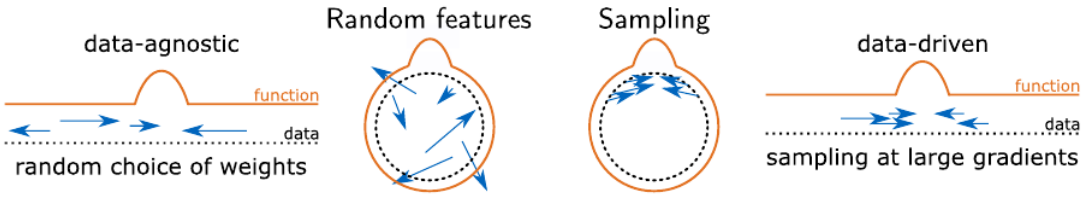


Figure 2.4: Illustration of weight placements in random feature models compared with sampling them where it matters: at large gradients, taken from [10].

The authors show that the SWIM sampled networks have better accuracy than random methods in various settings, from classification to transfer learning, and are comparable to trained networks while still outperforming training times even on a CPU. Additionally, universal approximation capabilities of SWIM sampled networks have been studied under the supervised setting to show that a SWIM sampled shallow network can approximate any continuous function on a compact subset of $\mathbb{R}^n$ uniformly well. Currently, GD-trained networks using backpropagation have much broader use cases and studies, e.g., SWIM networks are not studied for approximating vector fields; in this thesis, we aim to evaluate the performance of the SWIM sampled networks in Hamiltonian approximation.

Definition 2.3.1 (SWIM Sampled Neural Network [10])**.** Let Φ be a FFNN as defined in Def. 2.1.3. For $l = 1, \dots L$, let $(x_{0,i}^{(1)}, x_{0,i}^{(2)})_{i=1}^{N_l}$ be pairs of points sampled over $\mathcal{X} \times \mathcal{X}$. If the weights and biases of each layer $l = 1, 2, \dots, L$ and neuron $i = 1, 2, \dots, N_l$ have the form

$$w_{l,i} = s_1 \frac{x_{l-1,i}^{(2)} - x_{l-1,i}^{(1)}}{\left\| x_{l-1,i}^{(2)} - x_{l-1,i}^{(1)} \right\|^2}, \qquad b_{l,i} = \langle w_{l,i}, x_{l-1,i}^{(1)} \rangle + s_2 , \tag{2.13}$$

then Φ is a **SWIM sampled network**, where $\|\cdot\|$ is the L^2 norm, $\langle \cdot, \cdot \rangle$ the inner product, and

- $s_1, s_2 \in \mathbb{R}$ are constants to place the outputs of the activation function for the inputs $x^{(1)}$ and $x^{(2)}$. For tanh, the only activation function that we use in this thesis, $s_1 = 2s_2$ and $s_2 = \ln(3)/2$ are set, which implies $\sigma(x^{(1)}) = 1/2$ and $\sigma(x^{(2)}) = -1/2$, respectively, and $\sigma((x^{(1)} + x^{(2)})/2) = 0$,

- $x_{l-1,i}^{(k)} = \Phi^{(l-1)}(x_{0,i}^{(k)})$ for $k \in \{1,2\}$ and $x_{l-1,i}^{(1)} \neq x_{l-1,i}^{(2)}$,
- we write $w_{l,i}$ for the ith row of W_l and $b_{l,i}$ for the ith entry of b_l,
- for the parameters of the last (linear) layer we have $(W_{L+1}, b_{L+1}) = \underset{(W_{L+1},b_{L+1})}{\arg\min}\, \mathcal{L}(W_{L+1}\Phi^{(L)}(.) - b_{L+1})$, where $\mathcal{L}$ is the loss function we would like to minimize (e.g., can be solved using least squares solution as defined in Def. 2.1.2).

The authors also provide a probability distribution for the data points in the input space $\mathcal{X}$ (see Def. 2.3.2) to sample where it matters: at large gradients. The SWIM algorithm utilizes this distribution.

Definition 2.3.2 (Probability Density [10])**.** Let $\mathcal{X}$ be the input space and let $\mathcal{Y} = f(\mathcal{X})$ be the function space. For hidden layers $l = 1, \dots, L$, the **probability density** p_l^ϵ to sample pairs of points to be used in the SWIM sampling of the corresponding hidden layer (see Def. 2.3.1) can be defined through the proportionality

$$p_l^\epsilon\left(x_0^{(1)}, x_0^{(2)} | \{W_j, b_j\}_{j=1}^{l-1}\right) \propto \begin{cases} \frac{\left\|f(x_0^{(2)}) - f(x_0^{(1)})\right\|_{\mathcal{Y}}}{\max\left\{\left\|x_{l-1}^{(2)} - x_{l-1}^{(1)}\right\|_{\mathcal{X}_{l-1}}, \epsilon\right\}}, & \text{for } x_{l-1}^{(1)} \neq x_{l-1}^{(2)} \\ 0, & \text{otherwise} \end{cases}, \tag{2.14}$$

where

- $x_0^{(k)} \in \mathcal{X}$ and $x_{l-1}^{(k)} = \Phi^{(l-1)}(x_0^k)$ for $k \in \{1,2\}$ with the network $\Phi^{(l-1)}$ parameterized by SWIM sampled parameters: $\{W_j, b_j\}_{j=1}^{l-1}$, which can be defined recursively using the density function p_{l-1}^ϵ for $l > 1$,
- $\mathcal{X}_{l-1} = \Phi^{(l-1)}(\mathcal{X})$ is the output space of the $(l-1)$th layer,
- $\epsilon = 0$ for $l = 1$ and $\epsilon > 0$ otherwise. In this thesis, we set $\epsilon = 10^{-10}$ in all experiments,
- norms $\|\cdot\|_{\mathcal{Y}}$ and $\|\cdot\|_{\mathcal{X}_{l-1}}$ are arbitrary over their respective space, we choose the L^∞ norm for $\|\cdot\|_{\mathcal{Y}}$ and the L^2 norm for $\|\cdot\|_{\mathcal{X}_{l-1}}$ in this thesis for all experiments.

SWIM sampled networks can be constructed sequentially for each hidden layer using the above density given a set of data points and their function values.

Remark. The probability density in Def. 2.3.2 for sampling points in the input space can only be used in a supervised setting. When approximating PDEs, the training data is usually limited, i.e., we do not have access to the true function values. Bolager et

al. [10] mention providing an initial random guess, e.g., using uniform sampling over the input space and then using the approximate function values to utilize the SWIM algorithm. One of the methods we experiment with in section 3 is based on this idea.

3 Sampling Neural Networks to Approximate Hamiltonian Functions

In this section, we aim to approximate Hamiltonian functions using sampled networks. In section 3.1, we introduce a framework for constructing the sampled networks for Hamiltonian function approximation without the information of the true function values using physical laws as the inductive bias. In section 3.2, we evaluate the performance of the networks in various settings in the single pendulum (3.2.1), Lotka-Volterra (3.2.2), double pendulum (3.2.3), and Hénon-Heiles (3.2.4) systems. Lastly, in section 3.3, we show how the sampled networks can learn the Hamiltonian from trajectory data without the knowledge of gradients or vector fields using a numerical integrator and a post-training correction step to account for the discretization errors.

3.1 Framework for Learning Hamiltonian from Data Using Sampling

We propose a framework, called Random Hamiltonian NN (R-HNN), which leverages sampled networks to approximate target Hamiltonians from data (as defined in Def. 3.1.2). The sampled networks explained in section 2.3 need to be adapted for Hamiltonian approximation given limited data, where we do not have access to the true function values. The idea is similar to Eq. (2.8) designed for GP in [7]. We define our problem as in Def. 3.1.1.

Definition 3.1.1 (Hamiltonian Approximation Problem)**.** Given a target Hamiltonian $\mathcal{H} : \mathcal{X} \to \mathbb{R}$ on an Euclidean space ($\mathcal{X} \subseteq E \times E$, where $E = \mathbb{R}^d$) of a d DOF Hamiltonian system, we aim to find an approximation $\widehat{\mathcal{H}}$, where

$$\widehat{\mathcal{H}} = \min_{\widehat{\mathcal{H}}} \sum_{z \in \mathcal{Z}} \left\| \widehat{\mathcal{H}}(z) - \mathcal{H}(z) \right\|^2,$$

given a finite subset of the true phase space $\mathcal{Z} \subset \mathcal{X}$.

Definition 3.1.2 (Random Hamiltonian Neural Network (R-HNN))**.** Let Φ be an FFNN as defined in Def. 2.1.3, with input dimension $N_0 = D = 2d \geq 2$ and output dimension

$N_{L+1} = 1$, $\mathcal{H} : \mathcal{X} \to \mathbb{R}$ be the target d DOF Hamiltonian function that we would like to approximate, where $\mathcal{X} \subseteq E \times E$ and $E = \mathbb{R}^d$, and $\mathcal{D} = \{x_i, \dot{x}_i \,|\, x \in \mathcal{X} \wedge i \in [K]\} = \{q_i, p_i, \dot{q}_i, \dot{p}_i\}_{i=1}^K$ be the train set. Note that we assume we have no information about the true function values. Still, we would like to approximate the target function such that

$$\widehat{\mathcal{H}}(x) = \Phi(x) = W_{L+1}\Phi^{(L)}(x) - b_{L+1} \overset{!}{=} \mathcal{H}(x) \quad \text{for all } x \in \mathcal{D}, \tag{3.1}$$

where differentiating Eq. (3.1) w.r.t. x gives

$$\begin{aligned} \nabla\widehat{\mathcal{H}}(x) &\overset{!}{=} \nabla\mathcal{H}(x) \iff \\ \nabla\Phi(x) &\overset{!}{=} \nabla\mathcal{H}(x) \iff \\ \nabla\left(W_{L+1}\Phi^{(L)}(x) - b_{L+1}\right) &\overset{!}{=} \nabla\mathcal{H}(x) \iff \\ W_{L+1}\nabla\Phi^{(L)}(x) &\overset{!}{=} \nabla\mathcal{H}(x), \end{aligned} \tag{3.2}$$

where $x \in \mathcal{D}$. We aim to set up a system for all $x \in \mathcal{D}$. Since

$$\nabla\mathcal{H}(x) = (\nabla_q\mathcal{H}(x), \nabla_p\mathcal{H}(x))^T = \mathcal{J}^{-1}(\dot{q}, \dot{p})^T = (-\dot{p}, \dot{q})^T$$

with Hamilton's Eq. (2.4), we can use the vector fields from our train set $\dot{x} = (\dot{q}, \dot{p}) \in \mathcal{D}$ in the RHS of Eq. (3.2) to set up a linear system similar to Eq. (2.8), where we replace the GP ansatz with the NN ansatz and write

$$\begin{bmatrix} \nabla\widehat{\mathcal{H}}(x_1) \\ \nabla\widehat{\mathcal{H}}(x_2) \\ \vdots \\ \nabla\widehat{\mathcal{H}}(x_K) \\ \widehat{\mathcal{H}}(x_0) \end{bmatrix} = \underbrace{\begin{bmatrix} \nabla\Phi^{(L)}(x_1) \\ \nabla\Phi^{(L)}(x_2) \\ \vdots \\ \nabla\Phi^{(L)}(x_K) \\ \Phi^{(L)}(x_0) \end{bmatrix}}_{\in\mathbb{R}^{(2dK+1)\times N_L}} \cdot \underbrace{\left[W_{L+1}^T\right]}_{\in\mathbb{R}^{N_L}} \overset{!}{=} \underbrace{\begin{bmatrix} \mathcal{J}^{-1}\dot{x}_1 \\ \mathcal{J}^{-1}\dot{x}_2 \\ \vdots \\ \mathcal{J}^{-1}\dot{x}_K \\ \mathcal{H}(x_0) \end{bmatrix}}_{\in\mathbb{R}^{(2dK+1)}} \overset{2.4}{=} \begin{bmatrix} \nabla\mathcal{H}(x_1) \\ \nabla\mathcal{H}(x_2) \\ \vdots \\ \nabla\mathcal{H}(x_K) \\ \mathcal{H}(x_0) \end{bmatrix}, \tag{3.3}$$

where $\mathcal{J}^{-1} = \begin{bmatrix} 0_d & -I_d \\ I_d & 0_d \end{bmatrix}$ is the inverse of the matrix $\mathcal{J}$ (see Eq. (2.4)),

$$\mathcal{J}\mathcal{J}^{-1} = \begin{bmatrix} 0_d & I_d \\ -I_d & 0_d \end{bmatrix}\begin{bmatrix} 0_d & -I_d \\ I_d & 0_d \end{bmatrix} = \begin{bmatrix} I_d & 0_d \\ 0_d & I_d \end{bmatrix}.$$

We write $\nabla\Phi^{(L)}(x_i) \in \mathbb{R}^{2d\times N_L}$ for each point in the train set $x_i \in \mathcal{D}$ to the rows of the matrix on the left-hand-side (LHS) of Eq. (3.3) above, such that

$$\left[\nabla\Phi^{(L)}(x_i)\right] = \begin{bmatrix}\nabla_q\Phi^{(L)}(x_i)\\ \nabla_p\Phi^{(L)}(x_i)\end{bmatrix} = \underbrace{\begin{bmatrix}\nabla_{q_1}\Phi^{(L)}(x_i)\\ \nabla_{q_2}\Phi^{(L)}(x_i)\\ \vdots\\ \nabla_{q_d}\Phi^{(L)}(x_i)\\ \nabla_{p_1}\Phi^{(L)}(x_i)\\ \nabla_{p_2}\Phi^{(L)}(x_i)\\ \vdots\\ \nabla_{p_d}\Phi^{(L)}(x_i)\end{bmatrix}}_{\in\mathbb{R}^{2d\times N_L}},$$

where $\nabla_{q_j}(\cdot)$ is the jth entry of the gradient, corresponding to the partial derivative w.r.t. jth dimension of the input q. For p, $\nabla_{p_j}(\cdot)$ is written analogously. And we write $\mathcal{J}^{-1}\dot{x}_i$ and $\nabla\mathcal{H}(x_i)$ for each point in the train set $x_i \in \mathcal{D}$ on the RHS of Eq. (3.3), such that

$$\left[\mathcal{J}^{-1}\dot{x}_i\right] = \underbrace{\begin{bmatrix}-\dot{p}_i\\ \dot{q}_i\end{bmatrix}}_{\in\mathbb{R}^{2d}} = \left[\nabla\mathcal{H}(x_i)\right] = \begin{bmatrix}\nabla_q\mathcal{H}(x_i)\\ \nabla_p\mathcal{H}(x_i)\end{bmatrix} = \underbrace{\begin{bmatrix}\nabla_{q_1}\mathcal{H}(x_i)\\ \nabla_{q_2}\mathcal{H}(x_i)\\ \vdots\\ \nabla_{q_d}\mathcal{H}(x_i)\\ \nabla_{p_1}\mathcal{H}(x_i)\\ \nabla_{p_2}\mathcal{H}(x_i)\\ \vdots\\ \nabla_{p_d}\mathcal{H}(x_i)\end{bmatrix}}_{\in\mathbb{R}^{2d}},$$

where $\dot{q}_i$ and $\dot{p}_i$ are d-dimensional vectors since the system has d DOF. Finally, the last row $\Phi^{(L)}(x_0)$ is for fixing the integration constant, assuming that we know a real function value $\mathcal{H}(x_0)$ for a point $x_0 \in \mathcal{X}$. Since the last linear also includes a bias term b_{L+1} as in $\Phi(x_0) = W_{L+1}\Phi^{(L)}(x_0) - b_{L+1}$, we incorporate it by appending the column $[0,\dots,0,-1]^T$ of length $(2dK+1)$ to the matrix on the LHS of Eq. (3.3) and write

$$\underbrace{\begin{bmatrix}\nabla\Phi^{(L)}(x_1) & 0\\ \nabla\Phi^{(L)}(x_2) & 0\\ \vdots & \vdots\\ \nabla\Phi^{(L)}(x_K) & 0\\ \Phi^{(L)}(x_0) & 1\end{bmatrix}}_{A\in\mathbb{R}^{(2dK+1)\times(N_L+1)}} \cdot \underbrace{\begin{bmatrix}W_{L+1}^T\\ b_{L+1}\end{bmatrix}}_{w\in\mathbb{R}^{(N_L+1)}} \overset{!}{=} \underbrace{\begin{bmatrix}\mathcal{J}^{-1}\dot{x}_1\\ \mathcal{J}^{-1}\dot{x}_2\\ \vdots\\ \mathcal{J}^{-1}\dot{x}_K\\ \mathcal{H}(x_0)\end{bmatrix}}_{u\in\mathbb{R}^{(2dK+1)}}, \tag{3.4}$$

where the first $2dK$ entries for each data point are set to 0 because the bias in the last layer does not have any effect on the gradients (see Eq. (3.2)). The total number of network parameters can be calculated as

$$\underbrace{\sum_{l=1}^{L+1} N_{l-1} \cdot N_l}_{\text{weights}} + \underbrace{\sum_{l=1}^{L+1} N_l}_{\text{biases}} = \sum_{l=1}^{L+1} N_l(N_{l-1}+1). \tag{3.5}$$

For a shallow network, the number of parameters is $N_1(N_0+1) + N_2(N_1+1) = N_1(2d+2)+1$.
We reiterate:

- $\Phi^{(L)}(x_i)$ is the output of the final hidden layer given an input $x_i \in \mathcal{X}$,
- differentiating Eq. (3.1) w.r.t. the input x, we can derive Eq. (3.2) given a single data point $x \in \mathcal{X}$,
- if we stack Eq. (3.2) for each data point in our train set $x_i \in \mathcal{D}$, and incorporate Hamilton's equations, we get the linear system in Eq. (3.3),
- incorporating the term for an input $x_0 \in \mathcal{X}$ to fix the integration constant (for which we assume to know the true Hamiltonian $\mathcal{H}(x_0)$) together with the bias term in the last linear layer $b_{L+1} \in \mathbb{R}$, we get the final linear system in Eq. (3.4).

Fixing the hidden layers' weights and biases of Φ using any sampling scheme and then fitting the last (linear) layer's weights and bias of Φ by solving linear Eq. (3.4) reduces the number of parameters to be optimized (from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$, assuming $N_l \in \mathcal{O}(N)$ for some $N > 0$ for all layers $l \in [L+1]$). We use the linear least squares solution (see section 2.1.3) and make the optimization problem a convex optimization problem (see lemma 2.1.1), given as

$$\begin{bmatrix} W_{L+1}^T \\ b_{L+1} \end{bmatrix} = w = \arg\min_w \|Aw - u\|^2. \tag{3.6}$$

If a network is constructed using the above scheme, we call it an R-HNN. Note that we designed R-HNN specifically for the Hamiltonian approximation problem in Def. 3.1.1, and even though we numerically confirm in the next section 3.2 that we can reach low approximation errors for some Hamiltonians given a distinct test set $\mathcal{Z} \subset \mathcal{X}$, we do not provide any theoretical analysis here.

Remark. There is no constraint on the sampling scheme for the hidden layer weights and biases of Φ.

We use the framework defined in Def. 3.1.2 to define different types of R-HNNs, ones using data-agnostic (ELM in Def. 3.1.3) and the ones using data-driven (SWIM in Def. 3.1.4, 3.1.5, and 3.1.6) sampling schemes.

Definition 3.1.3 (ELM-R-HNN)**.** Let Φ be an R-HNN as defined in Def. 3.1.2. If the sampling scheme used for the weights and biases of the hidden layers is data-agnostic, then we call the network ELM-R-HNN. Solving a linear system to fit the last layer parameters has been extensively studied for ELM (see section 2.3.1). Therefore, we include "ELM" in the name of this specific network to specifically indicate that the model uses a data-agnostic RFM. For ELM-R-HNN, we use normal distribution $\mathit{Normal}(\mu, \sigma^2)$ with mean $\mu = 0$ and standard deviation $\sigma^2 = 1$ for sampling the hidden layer weights; and we use uniform distribution $\mathit{Uniform}(\min, \max)$ with interval $[\min, \max)$ for sampling the hidden layer biases. For all the experiments, we set the min and max to the minimum and maximum input range of $\{q_i, p_i\}_{i=1}^K$ in the train set, e.g. if the input range for q is sampled from -10 to 1, and for p is sampled from -1 to 10 then we set $\min = -10$ and $\max = 10$.

Remark. We do not fix min and max values in the uniform distribution used for sampling the biases in the hidden layers to have a more fair comparison of the weight sampling used in ELM between the other methods. However, suppose the sampling scheme is data-agnostic and domain-agnostic. Where we assume we have no prior knowledge about the domain of the problem, then we have to fix the min and max values for ELM without taking the domain or the data into account – e.g., Rahimi et al. [85] fix the bias to be picked uniformly from $[-\pi, \pi]$ for the RFMs they construct.

Definition 3.1.4 (SWIM-R-HNN)**.** Let Φ be an R-HNN as defined in Def. 3.1.2. If the network is also a SWIM sampled network according to the Def. 2.3.1, where the probability distribution in Def. 2.3.2 is used for sampling the points used in the parameter construction in Eq. (2.13), then we call the network SWIM-R-HNN. Important to note that the SWIM algorithm is a supervised sampling algorithm if the probability distribution in Def. 2.3.2 is used, where we assume to know the true function values. Therefore, for this architecture, the train set must additionally include the true function values $\mathcal{D} = \{x_i, \dot{x}_i, \mathcal{H}(x_i)\}_{i=1}^K$.

Definition 3.1.5 (U-SWIM-R-HNN)**.** Let Φ be an R-HNN as defined in Def. 3.1.2. If the network is also a SWIM sampled network according to Def. 2.3.1, where uniform probability distribution is used for sampling the points used in the parameter construction in Eq. (2.13), then we call the network Uniform-SWIM-R-HNN (U-SWIM-R-HNN).

Remark. Unlike SWIM-R-HNN (see Def. 3.1.4), we do not need the true function values for U-SWIM-R-HNN.

Definition 3.1.6 (A-SWIM-R-HNN)**.** Let Φ be an R-HNN as defined in Def. 3.1.2, and also be a SWIM sampled network according to Def. 2.3.1. Using the idea in remark 2.3.2, we first make an "initial guess" for the function values using U-SWIM-R-HNN and then use the approximate function values outputted by the U-SWIM-R-HNN $\{\widehat{\mathcal{H}}(x_i) = \Phi_{\text{U-SWIM}}(x_i)\}_{i=1}^{K}$ to construct SWIM-R-HNN. This removes the requirement of the true function values for the supervised SWIM algorithm, on the other hand, it requires the network to be fitted twice (first U-SWIM-R-HNN, then SWIM-R-HNN). To avoid confusion we name the final network as Approximate-SWIM-R-HNN (A-SWIM-R-HNN).

Algorithm 1: SWIM-R-HNN algorithm for constructing a SWIM-R-HNN (Def. 3.1.4). The hidden layer parameters are sampled according to the SWIM algorithm [10], and the last layer's parameters are computed using the least square solution of the linear system in Eq. (3.4).

Data: $\mathcal{D} = \{x_i, \dot{x}_i, \mathcal{H}(x_i) \mid x \in \mathcal{X} \wedge i \in [K]\}$, x_0, $\mathcal{H}(x_0)$

$\Phi^{(0)}(X) = X$;

for $l = 1, 2, \ldots, L$ **do**

  $W_l \in \mathbb{R}^{N_l \times N_{l-1}}, b_l \in \mathbb{R}^{N_l}$;

  $W_l, b_l \leftarrow$ `sampler`(SWIM) ;

end

compute $\nabla\Phi^{(L)}(X)$;

$W_{L+1}, b_{L+1} \leftarrow \arg\min_{w} \|Aw - u\|^2$;

return Φ;

The algorithm for constructing a SWIM-R-HNN is given in Alg. 1. Note the true function values $\mathcal{H}(X)$ that the SWIM sampler requires in this algorithm. In Alg. 2 the construction of other R-HNN methods (ELM, U-SWIM, and A-SWIM) are given. If the method is the A-SWIM method, the algorithm returns the output of Alg. 1 using the approximate function values outputted by the U-SWIM method. For both algorithms, the for loop realizes the sampling of the hidden layer parameters. Then, the gradients of the hidden layer outputs w.r.t. input are computed to construct the linear system in Eq. (3.4). This gradient can be computed analytically, e.g., for a shallow network given a point $x \in \mathcal{X}$ the hidden layer's output is

$$\Phi^{(L)}(x) = \sigma(W_1 x - b_1),$$

and if we differentiate the hidden layer's output w.r.t x, we get

Algorithm 2: R-HNN algorithm, for constructing ELM- (Def. 3.1.3), U-SWIM- (Def. 3.1.5), or A-SWIM-R-HNN (Def. 3.1.6). The hidden layer parameters are sampled according to either ELM- or U-SWIM-R-HNN and the last layer's parameters are computed using the least squares solution of the linear system in Eq. (3.4). If A-SWIM is specified, then the algorithm additionally calls SWIM-R-HNN (Alg. 1) using the approximate function values.

Data: $\mathcal{D} = \{x_i, \dot{x}_i \,|\, x \in \mathcal{X} \wedge i \in [K]\}$, x_0, $\mathcal{H}(x_0)$
method $\in \{\text{ELM}, \text{U-SWIM}, \text{A-SWIM}\}$

$\Phi^{(0)}(X) = X$;
for $l = 1, 2, \ldots, L$ **do**
  $W_l \in \mathbb{R}^{N_l \times N_{l-1}}, b_l \in \mathbb{R}^{N_l}$;
  if *method* is ELM **then**
    $W_l, b_l \leftarrow$ `sampler`(ELM) ;
  else
    $W_l, b_l \leftarrow$ `sampler`(U-SWIM) ;
  end
end
compute $\nabla\Phi^{(L)}(X)$;
$W_{L+1}, b_{L+1} \leftarrow \arg\min_w \|Aw - u\|^2$;
if *method* is A-SWIM **then**
  $\widehat{\mathcal{H}}(X) = \Phi(X)$;
  $\Phi \leftarrow$ `SWIM-R-HNN`$(\mathcal{D} \cup \widehat{\mathcal{H}}(X), x_0, \mathcal{H}(x_0))$;
end
return Φ;

$$\nabla\Phi^{(L)}(x) = \sigma'(W_1 x - b_1) \cdot W_1, \tag{3.7}$$

where σ' denotes the derivative of the activation function, and since we only use the tanh activation function ($\sigma = \tanh$) and it is differentiable, we have

$$\tanh'(\cdot) = \frac{\partial \tanh(\cdot)}{\partial(\cdot)} = 1 - \tanh^2(\cdot),$$

where $(\cdot)$ is used as placeholder for the term $W_1 x - b_1$ for simplicity. Because of this gradient computation, using a differentiable activation function is convenient for our framework. Also, note that we do not use automatic differentiation, as we can easily

implement the gradient computation using Eq. (3.7). Moreover, using the product rule, one can extend the gradient computation to support deep networks as well. For this thesis, however, we only experiment with shallow networks. After computing the gradient $\nabla\Phi^{(L)}(x)$, we then solve the linear system in Eq. (3.4) using the least squares solution in Eq. (3.6).

We do not provide any theoretical analysis (complexity, memory, etc.) regarding the Alg. 1 and Alg. 2 in this work. The algorithms are provided to the reader for explanation and do not fully align with the implementation decisions. Please see the code available at `https://github.com/AlphaGergedan/Random-HNN` for the exact implementation.

3.2 Numerical Experiments

We conduct the numerical experiments explained in this section on the Linux cluster segment CoolMUC-3 at Leibniz Supercomputing Centre (LRZ). CoolMUC-3 provides 148 nodes (64 cores per node) running at the nominal frequency of 1.3 GHz and with $\approx$ 96GB of memory (bandwidth 80.8 GB/s). Note that we only use a single node in all the experiments, as our algorithm is not designed for inter-node (distributed memory) parallelism yet.

For simplicity, we drop the R-HNN suffix from the names of the networks. As our baseline we use ELM to compare the data-agnostic random sampling against the data-driven sampling methods in all the experiments. For a fair evaluation, we train the networks multiple times (100 if not stated explicitly) with randomly sampled train $\mathcal{D} \subset \mathcal{X}$ and (distinct) test sets ($\mathcal{Z} \subset \mathcal{X} \wedge \mathcal{Z} \cap \mathcal{D} = \emptyset$) for each setting. Running multiple runs and taking the mean or median values when comparing instances, such as errors or training times, is necessary for such random models to make a fair comparison. We construct shallow networks with a single hidden layer with the tanh activation function for all the experiments and set the regularization constant of NumPy's `lstsq` to 10^{-13} when solving the linear optimization problem for the last layer parameters. For the SWIM methods, we also make sure that different pairs of points are selected for different neurons for the weight construction, i.e., $w_{l,i} \neq w_{l,j}$ for all $(i,j) \in [N_l]^2, i \neq j$ in Eq. (2.13). We do this by re-sampling if a duplicate pair is detected and repeating this until we get unique pairs, i.e., unique weights. Additionally, we plot the relative (rel.) L^2 error

$$\sqrt{\frac{\sum_i (\mathcal{H}(x_i) - \widehat{\mathcal{H}}(x_i))^2}{\sum_i \mathcal{H}(x_i)^2}}$$

when analyzing and evaluating the function approximations, where $x_i \in \mathcal{Z}$ if not

specified. We write $|\mathcal{D}|$ and $|\mathcal{Z}|$ for the train and test set size, respectively.

3.2.1 Single Pendulum

The Hamiltonian for the ideal single pendulum, also known as the nonlinear pendulum, is given as $\mathcal{H} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ with 1 degree of freedom. The Hamiltonian reads

$$\mathcal{H}(x) = \frac{p^2}{2ml^2} + mgl(1 - \cos(q)), \tag{3.8}$$

with its partial derivatives

$$\nabla_q \mathcal{H}(x) = mgl \sin(q)\,, \ \nabla_p \mathcal{H}(x) = \frac{p}{ml^2}, \tag{3.9}$$

where $q, p \in \mathbb{R}$ represent the angle and the angular momentum, respectively. An object with mass m is connected to a link of length l, and g is the gravitational acceleration. In this experiment, we set the constants to 1 ($m = l = g = 1$) and aim to approximate the Hamiltonian

$$\mathcal{H}(x) = \frac{p^2}{2} + (1 - \cos(q)) \tag{3.10}$$

for domains $[-2\pi, 2\pi] \times [-1, 1]$ and $[-2\pi, 2\pi] \times [-6, 6]$. The corresponding phase plots are depicted in Fig. 3.1, where we additionally plot the contour lines where the Hamiltonian is constant. One sees the oscillating behavior of the Hamiltonian due to the trigonometric term $\cos(\cdot)$ in Eq. (3.8).

We have fitted ELM, U-SWIM, A-SWIM, and SWIM variations of the R-HNN with $|\mathcal{D}|$ and network width scalings (see Fig. 3.2); keeping the network width at 1500 neurons for $|\mathcal{D}|$ scaling and keeping $|\mathcal{D}|$ at 10000 points for the network width scaling. Note again that the train set consists of points with their true derivative information $\mathcal{D} = \{x_i, \dot{x}_i\}_{i=1}^K$ for ELM, U-SWIM, and A-SWIM. SWIM additionally needs the true function values $\mathcal{H}(x_i)$ in the train set. For both experiments, $|\mathcal{Z}|$ is 10000, consisting of distinct points in the trained domain with their true function values $\mathcal{H}(\mathcal{Z})$.

We show that all the methods could reach very low approximation errors for the single pendulum system in Eq. (3.10). Especially in the larger domain and with smaller network widths, all the SWIM methods outperformed ELM. On the other hand, while the SWIM methods' performance remained consistent, ELM could reach very low approximation errors in both domains with large network widths (beyond 600 in the smaller and 2000 in the larger domain) and can outperform the other methods. With the larger domain, the gradients of the system move away from zero, especially in the momentum (p) direction till ∓ 6, whereas for the smaller domain, they only reach the values ∓ 1 (see the ground truth gradients in Appendix Fig. 3, and also in the Eq.

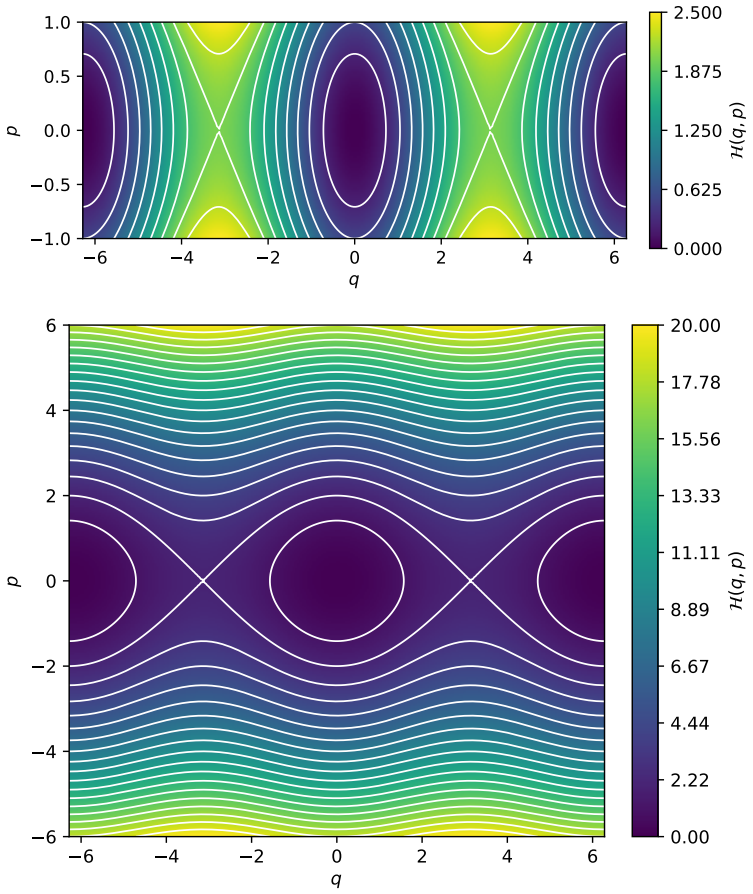


Figure 3.1: Ground truth phase plots of the single pendulum system in Eq. (3.10), domains $[-2\pi, 2\pi] \times [-1, 1]$ and $[-2\pi, 2\pi] \times [-6, 6]$ are displayed at the top and bottom respectively.

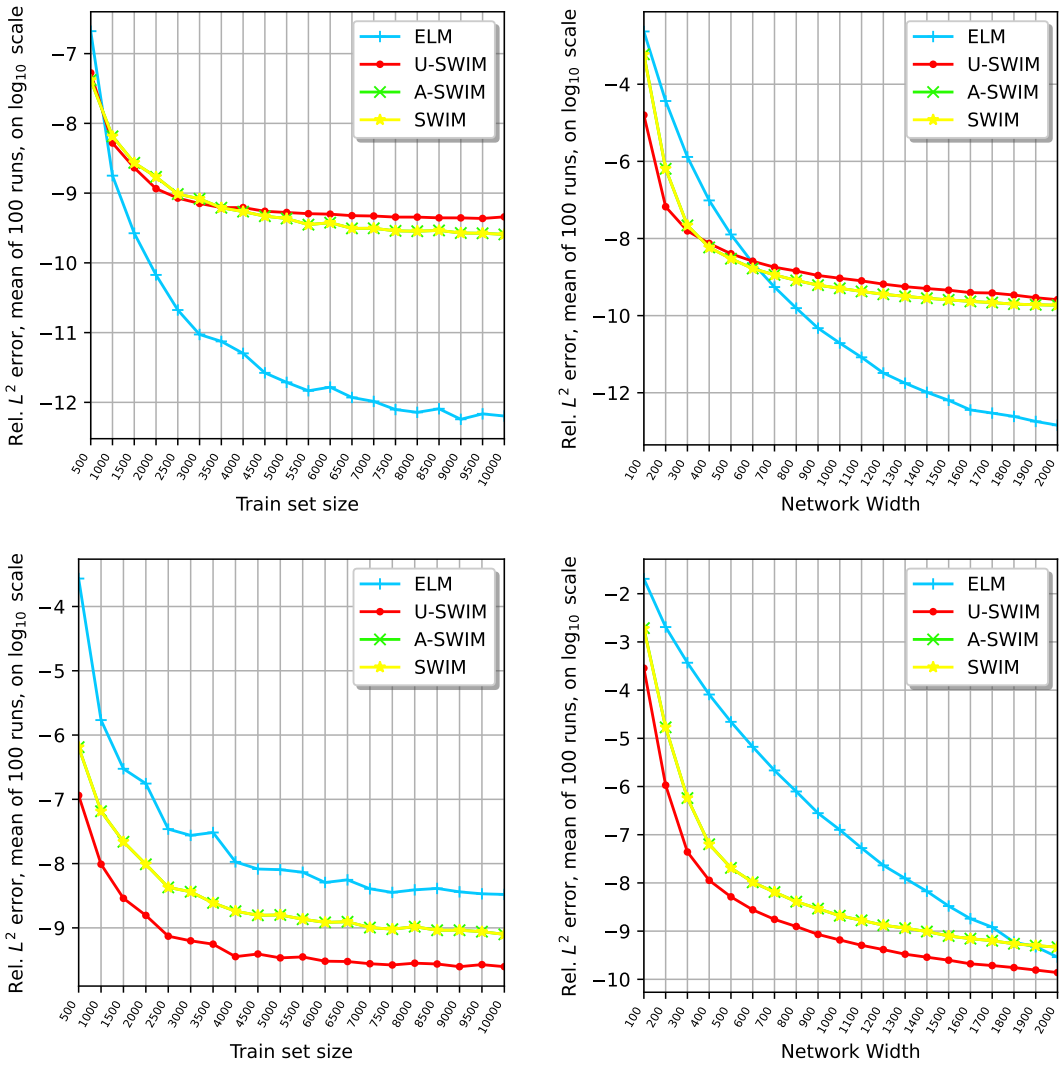


Figure 3.2: For single pendulum in Eq. (3.10) Hamiltonian errors (mean values) on $\mathcal{Z}$ with $|\mathcal{D}|$ and network width scaling are plotted. For $|\mathcal{D}|$ scaling, we fix the network width at 1500 neurons, and for the network width scaling, we set $|\mathcal{D}|$ to 10000. The plots at the top belong to the networks trained in domain $[-2\pi, 2\pi] \times [-1, 1]$, and the plots at the bottom belong to the networks trained in domain $[-2\pi, 2\pi] \times [-6, 6]$.

(3.9)). With large gradients, i.e., function values changing quickly, the SWIM methods can construct better weights using the data-driven scheme. Remarkably, A-SWIM can match the SWIM method's performance in all the settings at the time cost of retraining after an initial approximation (see Appendix Fig. 1 for time comparisons). Again, we emphasize that A-SWIM does not have access to the true function values but uses approximate function values to utilize the SWIM sampling scheme, whereas the SWIM here uses the true function values.

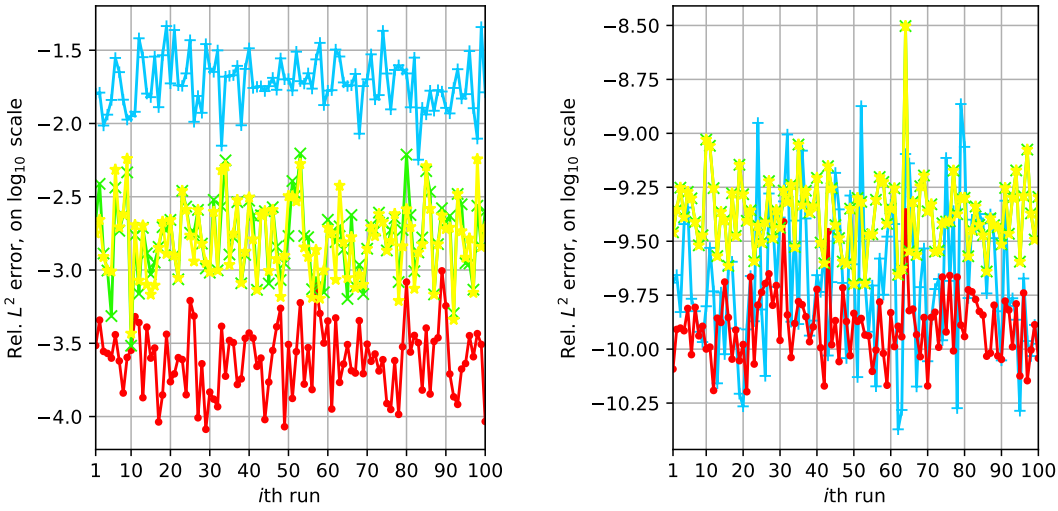


Figure 3.3: For single pendulum in Eq. (3.10) Hamiltonian errors on $\mathcal{Z}$ are plotted, where $|\mathcal{Z}| = 10000$. On the left, the networks have 100 neurons in the hidden layer, and on the right, 2000 neurons. Both setting has been trained with $|\mathcal{D}| = 10000$, and in domain $[-2\pi, 2\pi] \times [-6, 6]$.

In Fig. 3.3, we plot the errors on $\mathcal{Z}$ for each run with different network widths. The ELM method has strong oscillations with larger network widths or $|\mathcal{D}|$ (not shown here, see Appendix Fig. 2). A-SWIM cannot exactly follow the SWIM error curves when the approximation error is relatively higher in the left plot in Fig. 3.3. However, the difference is not noticeable if we take the mean or median error.

Additionally, we incorporate a frequency parameter f into the singular pendulum to experiment with different functions and modify the Hamiltonian as

$$\mathcal{H}(q,p) = \frac{p^2}{2} + (1 - \cos(fq)), \tag{3.11}$$

with its partial derivatives

$$\nabla_q \mathcal{H}(q,p) = f\sin(fq)\,,\ \nabla_p \mathcal{H}(q,p) = p. \tag{3.12}$$

Increasing f results in larger partial derivatives of the Hamiltonian w.r.t. q. The ground truth phase plots of the system with f set to 10, 15, and 20 can be found in Appendix Fig. 4.

In Fig. 3.4, we plot $|\mathcal{D}|$ scaling for different frequency parameters $f \in \{10, 15, 20\}$. The networks are trained with 1500 neurons in the hidden layer and tested with 20000 points in $\mathcal{Z}$ in domain $[\pi, \pi] \times [-1, 1]$. The gradients (position q part) get larger as the frequency parameter increases. When the gradients are larger, we report better approximations using the data-driven sampling schemes A-SWIM and SWIM. Again, A-SWIM follows the SWIM error curve quite accurately. Note that for the methods U-SWIM and ELM, adding more train samples does not change the error curves as they plateau very quickly, but the error curves of SWIM and A-SWIM continue to decrease. This may indicate that having more points in $\mathcal{D}$ affects the weight sampling scheme in the SWIM and A-SWIM methods, effectively sampling better parameters. Network width scaling with $|\mathcal{D}| = 10000$ is also included in Appendix Fig. 5 for reference with similar results. We also plot errors for each run to analyze the consistency of the methods in Fig. 3.5. The ELM error curve does not oscillate as much as in the previous experiment (compared to Fig. 3.3). The SWIM methods consistently oscillate in the rel. L^2 error order of ∓ 0.5. Note that we did not include the gradient errors on $\mathcal{D}$ and $\mathcal{Z}$ as they match the Hamiltonian error curves on $\mathcal{Z}$. So, for this experiment, one can say that all the models could generalize well: The performance on approximating the gradients of the Hamiltonian is comparable to the Hamiltonian function approximation.

3.2.2 Lotka-Volterra

The Hamiltonian of the Lotka-Volterra system [31] with 1-DOF is given as $\mathcal{H} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$. The Hamiltonian reads

$$\mathcal{H}(q,p) = \beta e^q - \alpha q + \delta e^p - \gamma p \tag{3.13}$$

with its partial derivatives

$$\nabla_q \mathcal{H}(q,p) = \beta e^q - \alpha\,,\ \nabla_p \mathcal{H}(q,p) = \delta e^p - \gamma, \tag{3.14}$$

where $q, p \in \mathbb{R}$ represent the prey and predator population densities, respectively. We set the parameters as $\beta = -1$, $\alpha = -2$, $\delta = -1$, $\gamma = -1$. This lets us experiment with

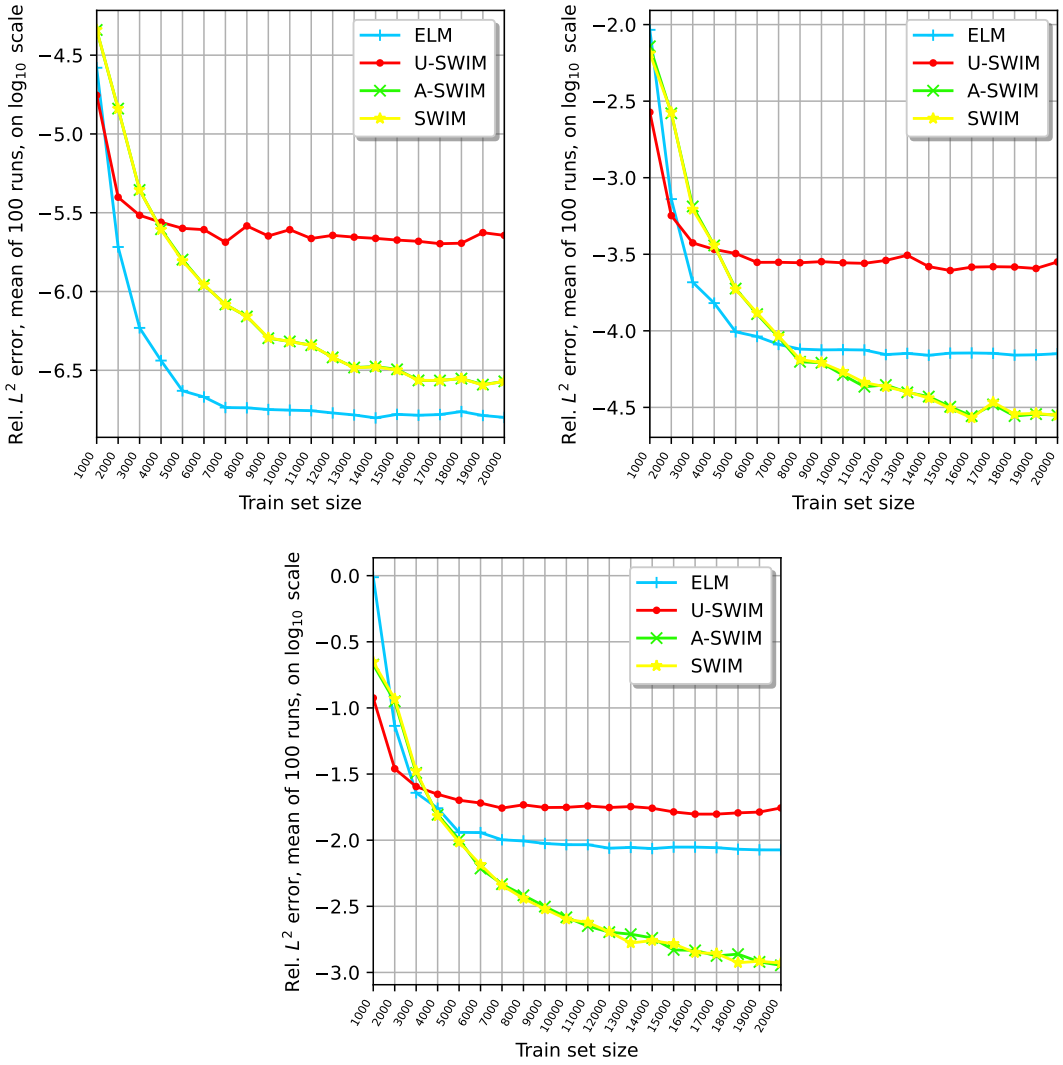


Figure 3.4: For single pendulum in Eq. (3.11) Hamiltonian errors on $\mathcal{Z}$ are plotted. All the settings are trained in domain $[\pi, \pi] \times [-1, 1]$ with 1500 network width. $|\mathcal{D}|$ scaling is plotted for different settings. Frequency (f) is set to 10 at the top left, 15 at the top right, and 20 at the bottom plot.

an equilibrium point near the zero vector, as we have

$$\nabla_q \mathcal{H}(q, p) = 0 \iff q = \ln(\alpha/\beta) \approx 0.69,$$
$$\nabla_p \mathcal{H}(q, p) = 0 \iff p = \ln(\gamma/\delta) = 0,$$

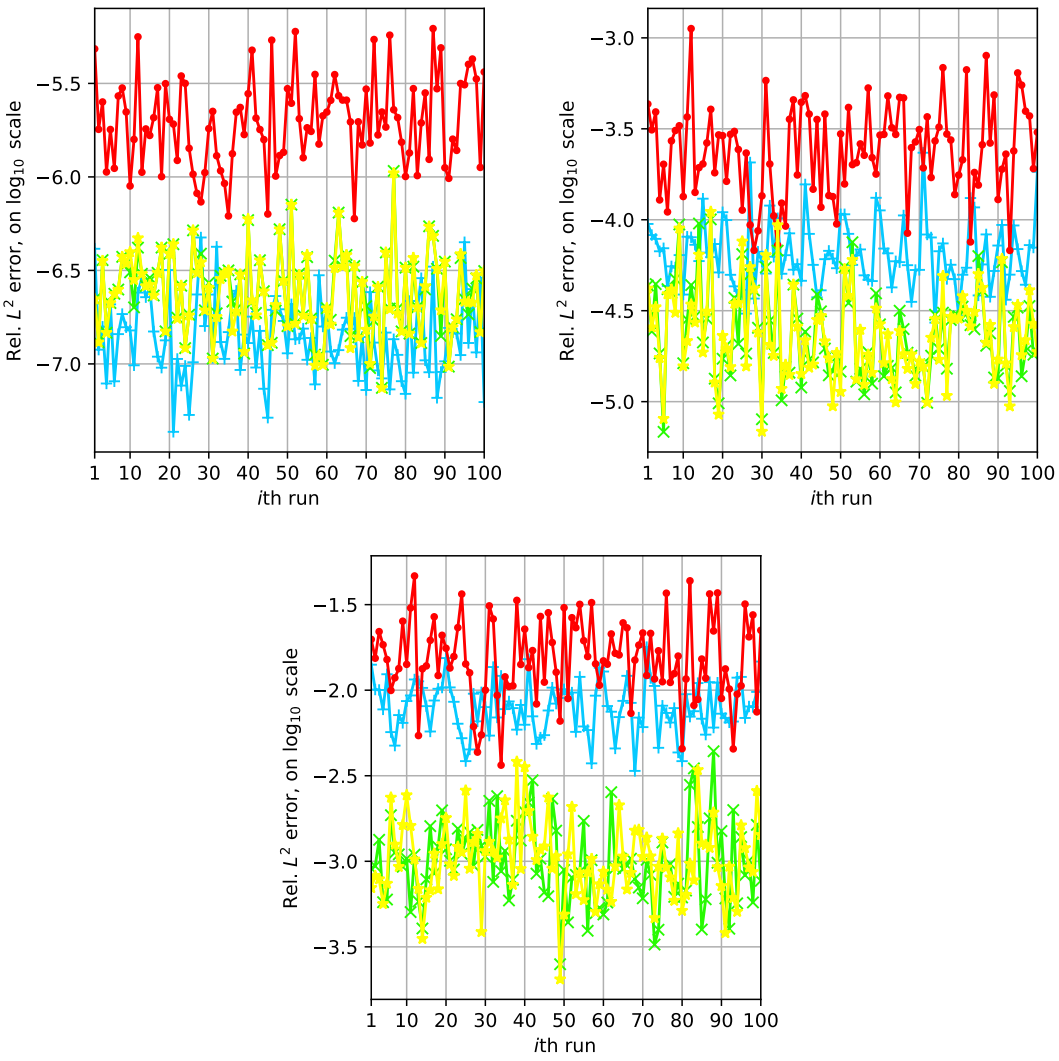


Figure 3.5: For single pendulum in Eq. (3.11) Hamiltonian errors on $\mathcal{Z}$ are plotted. All the settings are trained in domain $[\pi, \pi] \times [-1, 1]$ with 1500 network width and $|\mathcal{D}| = 20000$. Rel. L^2 error in each run is plotted for different settings. Frequency (f) is set to 10 at the top left, 15 at the top right, and 20 at the bottom plot.

resulting in the Hamiltonian

$$\mathcal{H}(q, p) = -e^q + 2q - e^p + p, \tag{3.15}$$

which we aim to approximate for domains $[-2,2]^2$ and $[-5,5]^2$.

We also experiment with the parameters set as $\alpha = 3.5$, $\beta = 0.025$, $\delta = 0.07$ and $\gamma = 10$. In this setting, the system has an equilibrium point close to 5 in both dimensions:

$$\nabla_q \mathcal{H}(q,p) = 0 \iff q = \ln(\alpha/\beta) \approx 4.94,$$
$$\nabla_p \mathcal{H}(q,p) = 0 \iff p = \ln(\gamma/\delta) \approx 4.96,$$

resulting in the Hamiltonian

$$\mathcal{H}(q,p) = 0.025e^q - 3.5q + 0.07e^p - 10p, \tag{3.16}$$

which we aim to approximate for domain $[0,8]^2$. In Fig. 3.6, the ground truth phase plots are depicted for the settings we would like to approximate.

Again, we experiment with different numbers of neurons in the hidden layer. The Hamiltonian approximation errors are plotted in Fig. 3.7 for the three systems we would like to approximate. $|\mathcal{D}| = |\mathcal{Z}| = 10000$ for all the settings in this experiment. Compared to SWIM methods, ELM performs again better with larger network widths for approximating Eq. (3.15); however, it performs worse for approximating Eq. (3.16) even with large network widths. Like the single pendulum system, ELM performed worse with smaller network widths and larger domain $[-5,5]^2$. Again, in the larger domain, the gradients are bigger (see Appendix Fig. 6 and Appendix Fig. 7).

In Fig. 3.8 and 3.9, we plot the gradient error on $\mathcal{D}$ and $\mathcal{Z}$ respectively. The gradient error on $\mathcal{D}$ represents the objective that we minimize in the linear system in Eq. (3.4), and the gradient error on $\mathcal{Z}$ represents the generalized performance, i.e., how good it performs on unseen data. For Eq. (3.15), we see that the curves of the gradient errors on $\mathcal{D}$ and $\mathcal{Z}$ match (the plots on the left and at the center). They also match the real objective we want to minimize (the Hamiltonian error on $\mathcal{Z}$). For Eq. (3.16), however, this is not the case for ELM. While the gradient errors on $\mathcal{D}$ can reach 10^{-10} rel. L^2 errors, the model does not generalize well on the test set $\mathcal{Z}$ and can only reach around 10^{-8} rel. L^2 errors like the SWIM methods. Remarkably, while ELM approximates the gradients quite as accurately as the SWIM methods, it cannot approximate the Hamiltonian as accurately as the SWIM methods on $\mathcal{Z}$. Increasing the network width above 2000 also does not change the results, which we did not include here. This may indicate that the ELM method is overfitting in this case. We also include the plots where we have scaled $|\mathcal{D}|$ and got similar results (see Appendix Fig. 8, Appendix Fig. 9, and Appendix Fig. 10). This was not the case with the single pendulum system in the previous experiments, where all the models could generalize well.

Regarding the consistency of the methods, we also analyze the oscillations in the error curves. Like in the single pendulum experiments, the SWIM methods' error

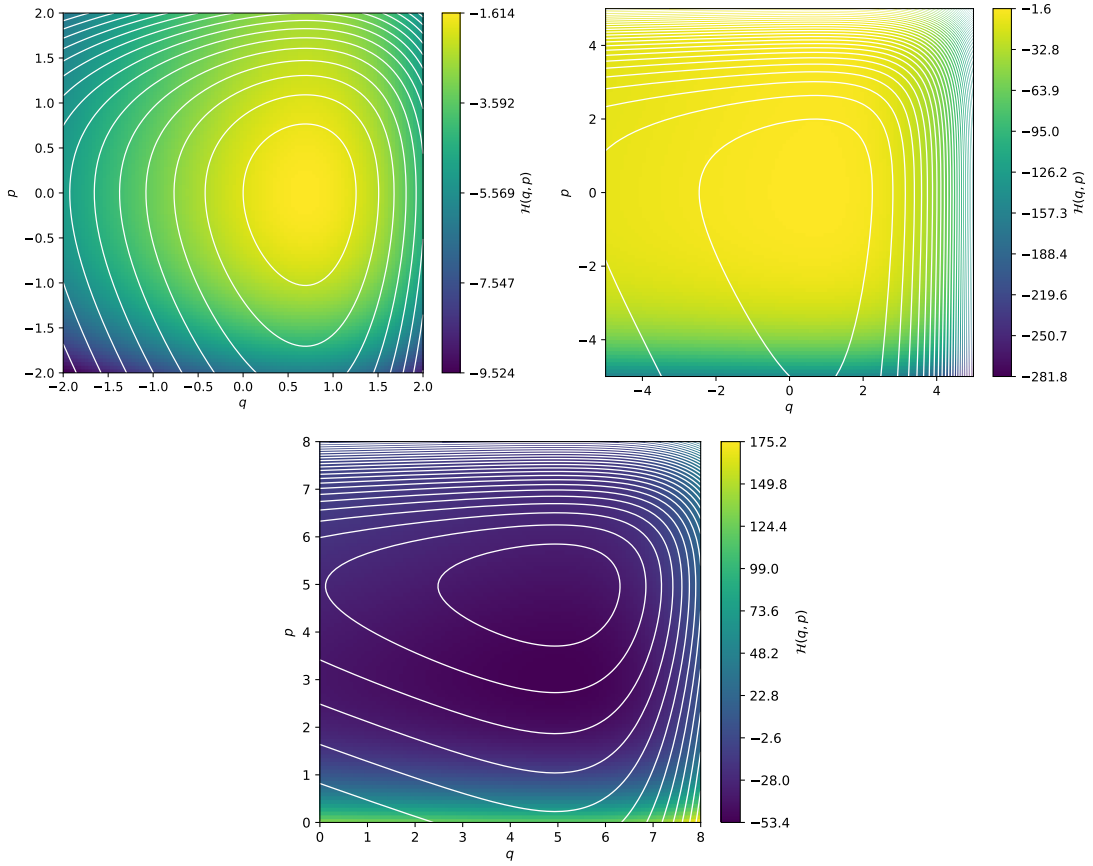


Figure 3.6: Ground truth phase plots of the Lotka-Volterra system. The plots at the top left and top right represent Eq. (3.15) in domains $[-2,2]^2$ and $[-5,5]^2$, respectively. The plot at the bottom represents Eq. (3.16) in domain $[0,8]^2$.

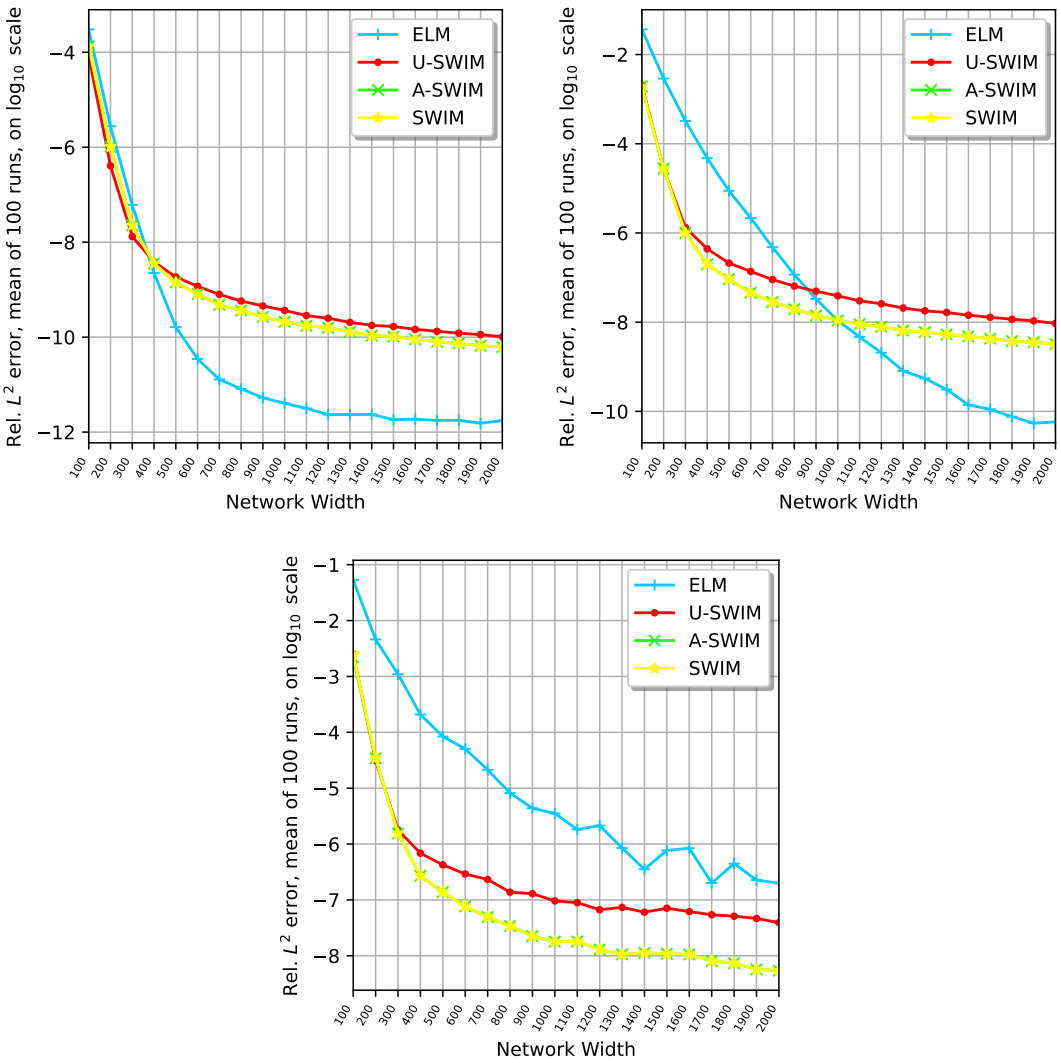


Figure 3.7: Lotka-Volterra Hamiltonian errors on $\mathcal{Z}$ are plotted, and $|\mathcal{D}| = 10000$. The plots at the top left and top right are zero-centered, as in Eq. (3.15), and are trained in domains $[-2,2]^2$ and $[-5,5]^2$ respectively. The plot at the bottom is centered around $(5,5)$, as in Eq. (3.16), and is trained in domain $[0,8]^2$.

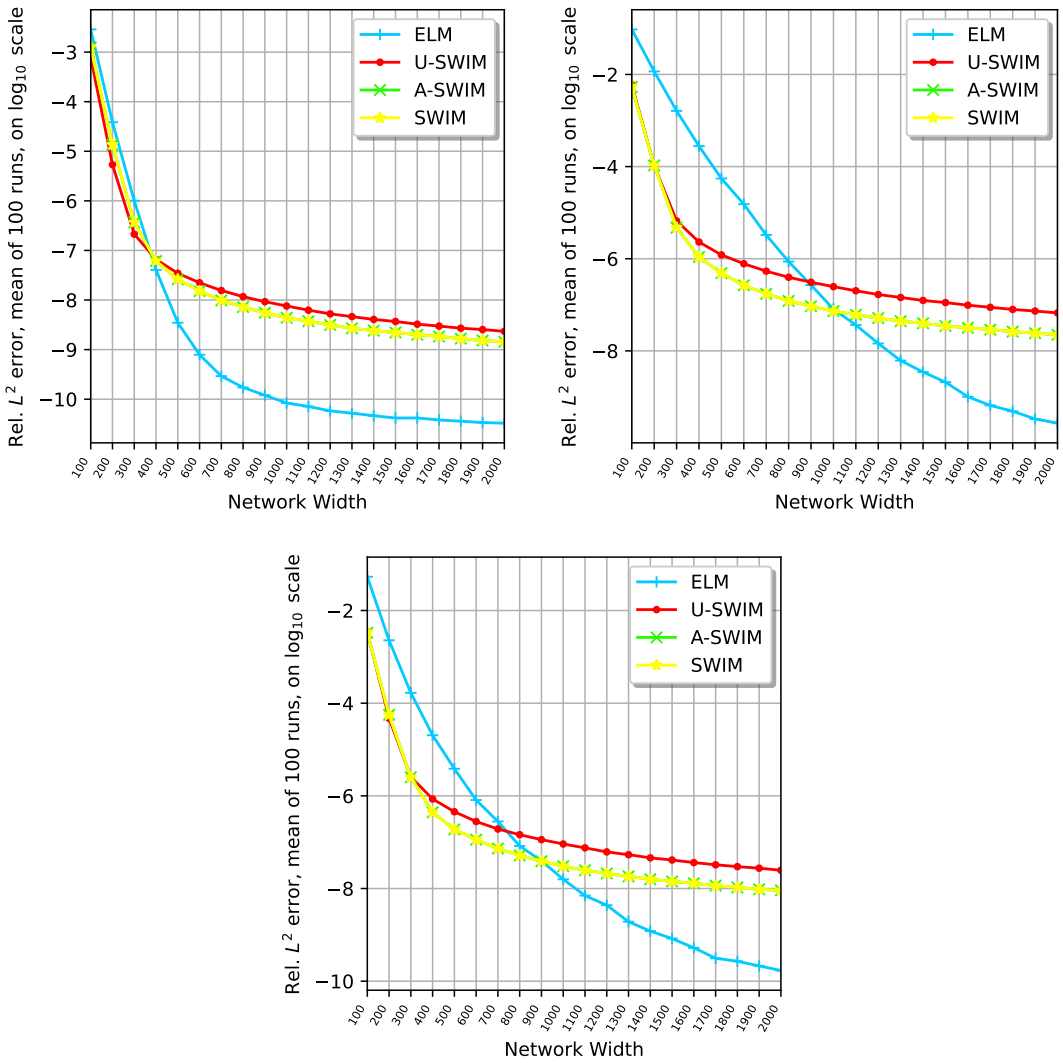


Figure 3.8: Lotka-Volterra gradient errors on $\mathcal{D}$ are plotted, and $|\mathcal{D}| = 10000$. The plots at the top left and top right are zero-centered, as in Eq. (3.15), and are trained in domains $[-1,1]^2$ and $[-5,5]^2$ respectively. The plot at the bottom is centered around $(5,5)$, as in Eq. (3.16), and is trained in domain $[0,8]^2$.

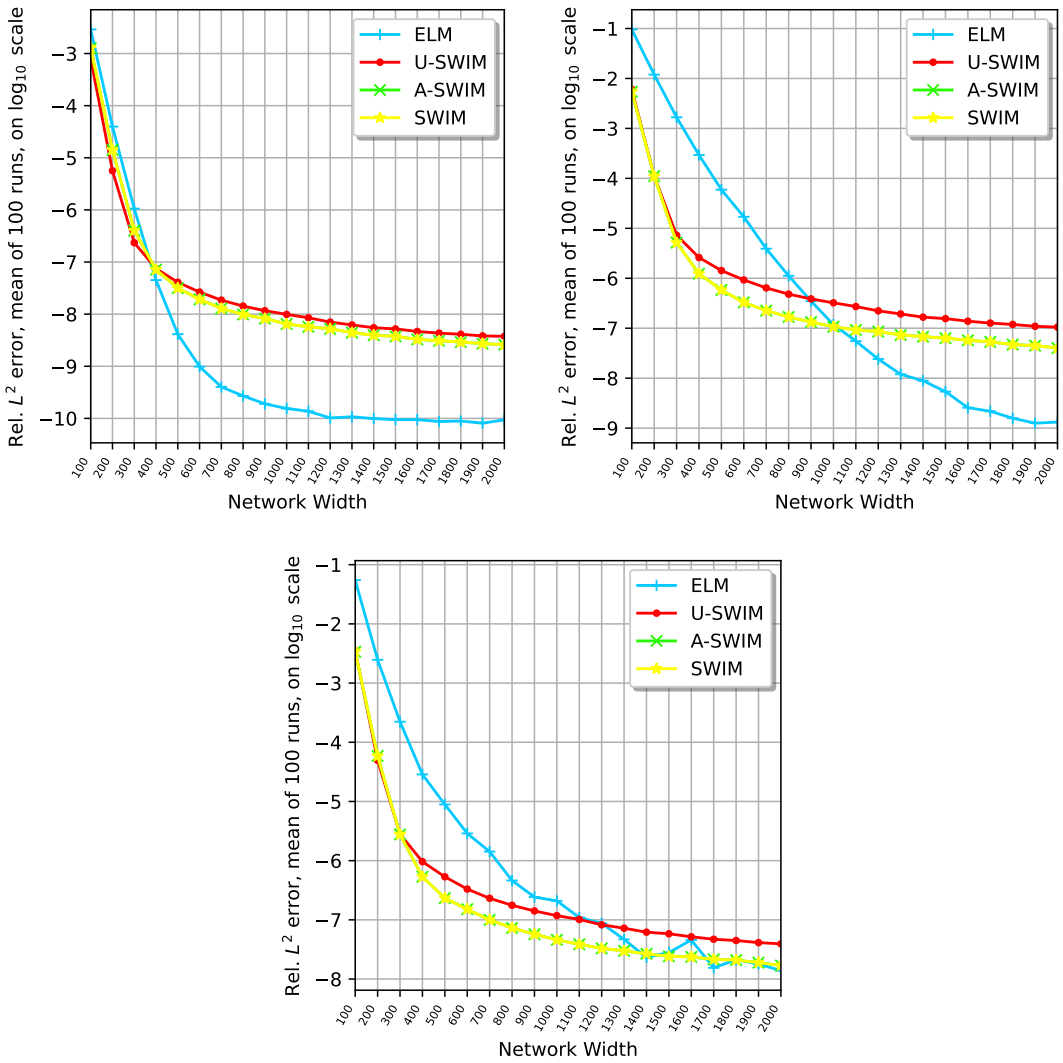


Figure 3.9: Lotka-Volterra gradient errors on $\mathcal{Z}$ are plotted, and $|\mathcal{D}| = 10000$. The plots at the top left and top right are zero-centered, as in Eq. (3.15) trained in domains $[-1,1]^2$ and $[-5,5]^2$ respectively. The plot at the bottom is centered around $(5,5)$ (Eq. (3.16)) trained in domain $[0,8]^2$.

curves are more consistent. In contrast, the ELM error curve oscillates with a much larger frequency with large network widths (see Appendix Fig. 11 and Appendix Fig. 12).

3.2.3 Double Pendulum

The Hamiltonian of the higher-dimensional, chaotic, and non-separable, double pendulum system with 2-DOF is given as $\mathcal{H} : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$. In the system, an object with mass m_1 is connected to a link of length l_1, and g is the gravitational acceleration. Additionally, another object with mass m_2 is connected through a link of length l_2 to the first object. The Hamiltonian reads

$$\begin{aligned}\mathcal{H}(x) = &\frac{m_2 l_2^2 p_1^2 + (m_1 + m_2) l_1^2 p_2^2 - 2 m_2 l_1 l_2 p_1 p_2 \cos(q_1 - q_2)}{2 m_2 l_1^2 l_2^2 (m_1 + m_2 \sin^2(q_1 - q_2))} \\ &- (m_1 + m_2) g l_1 \cos(q_1) - m_2 g l_2 \cos(q_2),\end{aligned} \tag{3.17}$$

with its partial derivatives

$$\begin{aligned}\nabla_{q_1} \mathcal{H}(x) &= (m_1 + m_2) g l_1 \sin(q_1) + h_1 - h_2 \sin(2(q_1 - q_2)), \\ \nabla_{q_2} \mathcal{H}(x) &= m_2 g l_2 \sin(q_2) - h_1 + h_2 \sin(2(q_1 - q_2)), \\ \nabla_{p_1} \mathcal{H}(x) &= \frac{l_2 p_1 - l_1 p_2 \cos(q_1 - q_2)}{l_1^2 l_2 (m_1 + m_2 \sin^2(q_1 - q_2))}, \\ \nabla_{p_2} \mathcal{H}(x) &= \frac{(m_1 + m_2) l_1 p_2 - m_2 l_2 p_1 \cos(q_1 - q_2)}{m_2 l_1 l_2^2 (m_1 + m_2 \sin^2(q_1 - q_2))},\end{aligned} \tag{3.18}$$

where $x = (q, p)$, $q = (q_1, q_2) \in \mathbb{R}^2$ represent the angles, and $p = (p_1, p_2) \in \mathbb{R}^2$ represent the angular momenta of the objects. We write $\nabla_{q_i} \mathcal{H}(x)$ for $\frac{\partial \mathcal{H}(x)}{\partial q_i}$ and $\nabla_{p_i} \mathcal{H}(x)$ for $\frac{\partial \mathcal{H}(x)}{\partial p_i}$ for $i \in \{1, 2\}$, and h_1 and h_2 are defined as

$$\begin{aligned}h_1 &= \frac{p_1 p_2 \sin(q_1 - q_2)}{l_1 l_2 (m_1 + m_2 \sin^2(q_1 - q_2))}, \\ h_2 &= \frac{m_2 l_2^2 p_1^2 + (m_1 + m_2) l_1^2 p_2^2 - 2 m_2 l_1 l_2 p_1 p_2 \cos(q_1 - q_2)}{2 l_1^2 l_2^2 (m_1 + m_2 \sin^2(q_1 - q_2))^2}.\end{aligned}$$

In this experiment, we set the constants to 1 ($m_1 = m_2 = l_1 = l_2 = g = 1$) and aim to approximate the resulting Hamiltonian

$$\mathcal{H}(x) = \frac{p_1^2 + 2p_2^2 - 2p_1p_2\cos(q_1 - q_2)}{2(1 + \sin^2(q_1 - q_2))} - 2\cos(q_1) - \cos(q_2), \tag{3.19}$$

in domain $[-\pi, \pi] \times [-1, 1]$.

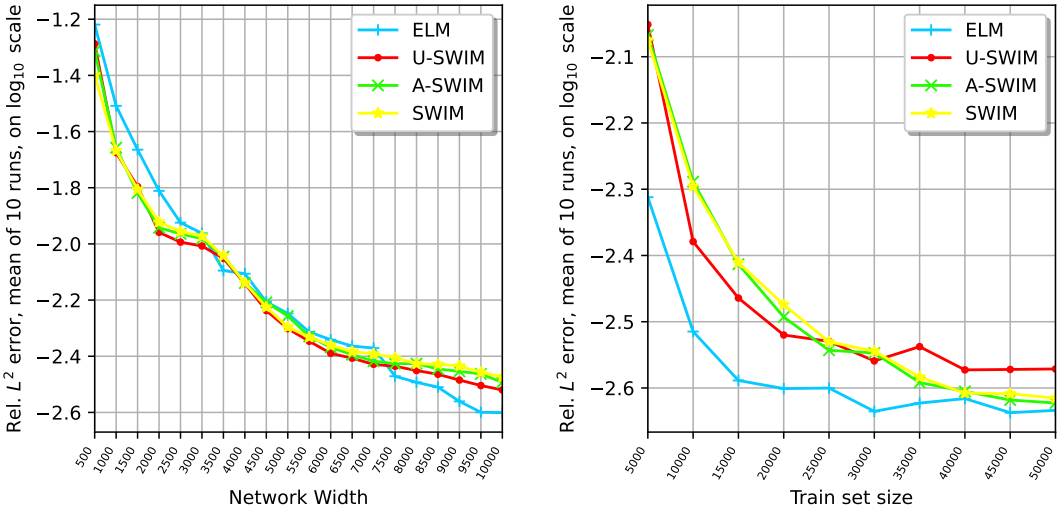


Figure 3.10: For double pendulum in Eq. (3.19) Hamiltonian errors on $\mathcal{Z}$ are plotted, where $|\mathcal{Z}| = 20000$. The left plot is network width scaling with $|\mathcal{D}| = 20000$. The right plot is $|\mathcal{D}|$ scaling with network width set to 10000. Networks in both settings are trained in domain $[-\pi, \pi] \times [-1, 1]$.

In Fig. 3.10, mean Hamiltonian errors on $\mathcal{Z}$ are plotted. For this experiment, we took the mean error values of 10 runs for each setting, instead of 100 runs as in previous experiments. Low approximation errors (≈ 0.002 rel. L^2 error) can again be reached in this experiment for the double pendulum system using all the methods. Although the difference is small, the ELM method performs better in larger network widths. In $|\mathcal{D}|$ scaling, similar to the single pendulum frequency scaling plots in Fig. 3.4, ELM and U-SWIM methods seem to plateau a little more quickly than the A-SWIM and SWIM methods. Like in all the previous experiments, the A-SWIM error curve again follows the SWIM error curve quite accurately in both plots. In Appendix Fig. 13 and Appendix Fig. 14, one can find the corresponding gradient error plots on $\mathcal{D}$ and the train times, respectively.

We have integrated the learned dynamics using the trained networks' gradients to demonstrate the networks' conserved quantities. We have previously evaluated the approximation performance of the networks by comparing the mean values of the rel.

L^2 errors of multiple runs; here we take the model with the median rel. L^2 error instead, as we cannot realize the model with the mean error value. Note that learning the flow map was never our goal. We use the explicit implementation of the symplectic Euler integrator for its simplicity, which, to be noted, is not symplectic in this case because the Hamiltonian of the double pendulum is not separable (see numerical integrators section 2.1.2 for more explanation). Still, we can confirm by using a small time step that the Hamiltonian of the learned systems matches the real Hamiltonian of the system. In Fig. 3.11, we plot one of the corresponding integration results, with errors on the predicted trajectory and the conservation of the Hamiltonian over time. For integration and Poincaré plots, see Appendix Fig. 15 and Appendix Fig. 16 respectively. Fig. 3.12 shows similar results using a different initial condition. For integration and Poincaré plots, see Appendix Fig. 17 and Appendix Fig. 18. One can see that different initial conditions can have different outcomes for the methods. This can be because SWIM methods better approximated the Hamiltonian in some parts of the domain; however, the methods generally showed similar performance. An important remark is that we demonstrate that the networks can preserve the Hamiltonian of the system quite accurately.

3.2.4 Hénon-Heiles

Another higher-dimensional and chaotic system with 2-DOF is the Hénon-Heiles [51], given as $\mathcal{H} : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$. The Hamiltonian reads

$$\mathcal{H}(x) = \frac{1}{2}(p_1^2 + p_2^2) + \frac{1}{2}(q_1^2 + q_2^2) + \alpha(q_1^2 q_2 - \frac{1}{3}q_2^3), \tag{3.20}$$

with its partial derivatives

$$\begin{aligned} \nabla_{q_1}\mathcal{H}(x) &= q_1 + 2\alpha q_1 q_2, \\ \nabla_{q_2}\mathcal{H}(x) &= q_2 + \alpha(q_1^2 - q_2^2), \\ \nabla_{p_1}\mathcal{H}(x) &= p_1, \\ \nabla_{p_2}\mathcal{H}(x) &= p_2, \end{aligned} \tag{3.21}$$

where $x = (q, p)$, $q = (q_1, q_2) \in \mathbb{R}^2$ represent the coordinates, $p = (p_1, p_2) \in \mathbb{R}^2$ represent the corresponding momenta in a 2-dimensional plane, and α is a bifurcation parameter. The system represents the motion of a star around a galactic center. Unlike the double pendulum, the Hénon-Heiles system is separable.

We set $\alpha \in \{0, 0.5, 0.7, 0.9, 1\}$ and aim to approximate the resulting Hamiltonians from Eq. (3.20) in domain $[-1, 1]^2$. Fig. 3.13 shows the mean Hamiltonian errors for α

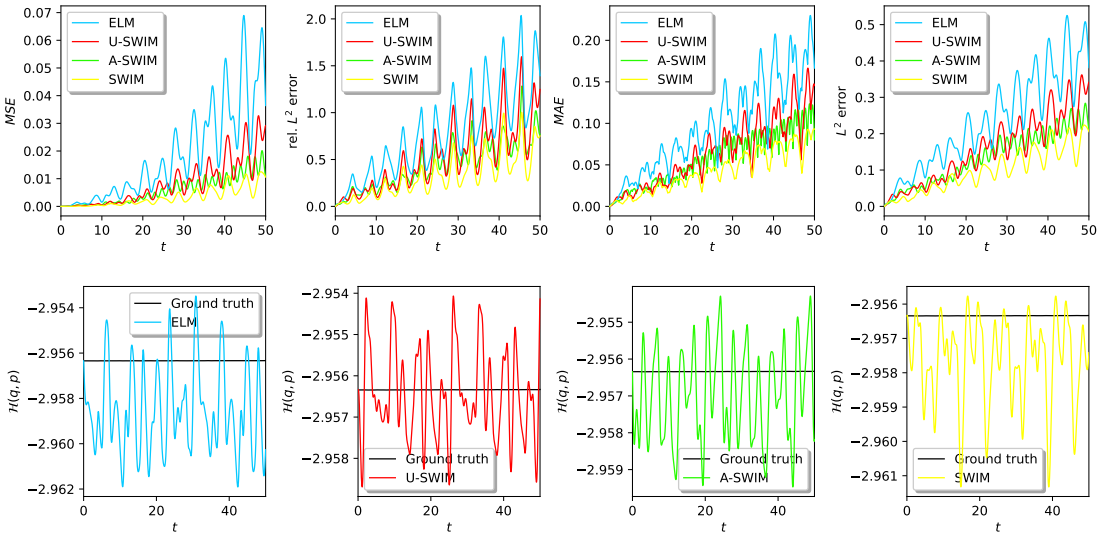


Figure 3.11: For double pendulum in Eq. (3.19) trajectory errors and Hamiltonian values are plotted in the top and bottom row respectively, integrated with $h = 10^{-4}$. The median model is used (with the median approximation error on the rel. L^2 error), with 10000 neurons trained with $|\mathcal{D}| = 20000$. The Initial value is $(0.15, 0.1, -0.05, 0.1)$. MSE is the mean squared error, and MAE is the mean absolute error. The ground truth in the bottom row represents the symplectic Euler using the true Hamiltonian.

set to 0 and 1; other settings produce similar results and thus are not shown here. Like the double pendulum experiment, we display the mean error values of 10 runs for each setting. The important remark is that some partial derivatives in Eq. (3.21) scale with α. Therefore, the function may undergo quicker changes for some components for higher α values, e.g., $\partial\mathcal{H}(x)/\partial q_1$ can be at most 3 in domain $[-1,1]^2$ for $\alpha = 1$, whereas it can be at most 1 for $\alpha = 0$. The resulting change in the partial derivatives when α is set to 1 instead of 0 is not big, but it may slightly affect the performance of the ELM method, as seen in Fig. 3.13. In contrast, the SWIM methods follow the same approximation error curves. The corresponding gradient errors on $\mathcal{D}$ and $\mathcal{Z}$ and the Hamiltonian errors on $\mathcal{D}$ are similar to Fig. 3.13 and, therefore, are omitted here. See Appendix Fig. 19 for corresponding gradient errors on $\mathcal{D}$ and the training times.

Like the double pendulum experiment, we integrate the learned dynamics of the Hénon-Heiles approximations using the median model. We again use the explicit implementation of the symplectic Euler, which becomes symplectic for the Hénon-Heiles system as it is separable. We only plot the system with $\alpha = 1$ here for demonstration,

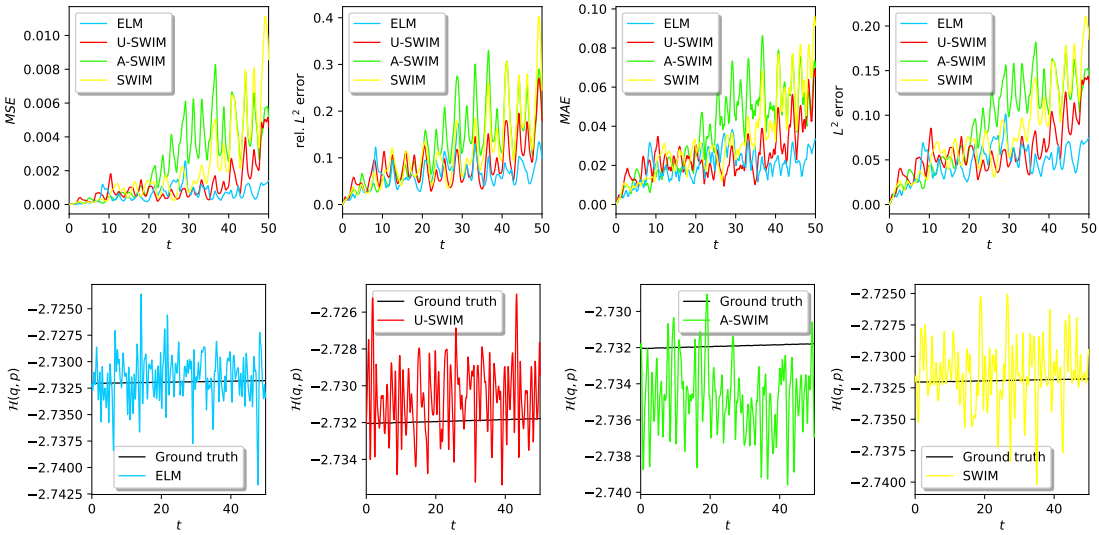


Figure 3.12: Same settings as in Fig. 3.11, but with the initial value $(\pi/6, 0, 0, 0)$. Again, the ground truth in the bottom row represents the symplectic Euler using the true Hamiltonian.

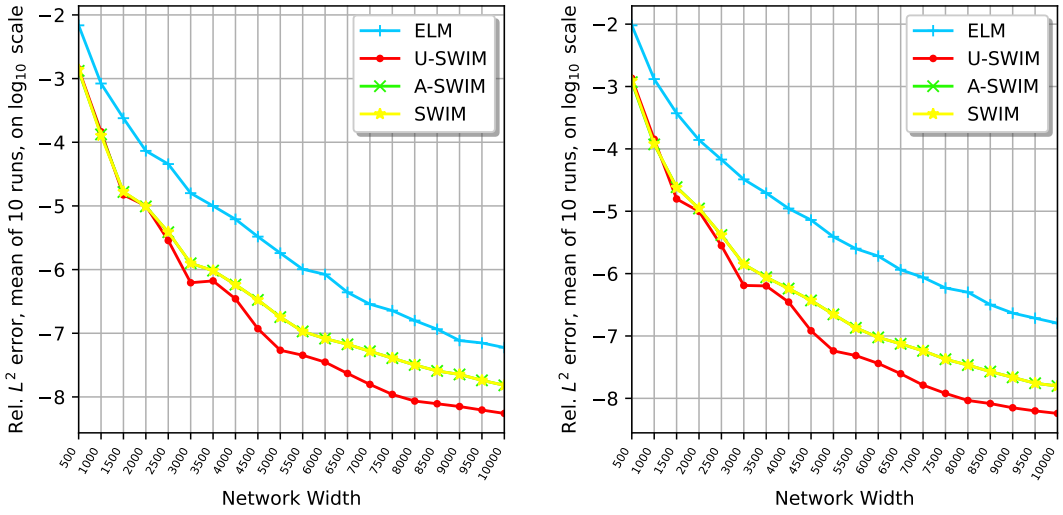


Figure 3.13: For Hénon-Heiles in Eq. (3.20) Hamiltonian errors on $\mathcal{Z}$ are plotted, where $|\mathcal{Z}| = 20000$. Network width scaling is displayed with $|\mathcal{D}| = 20000$. Networks are trained in domain $[-1, 1]^2$.

which should produce a chaotic behavior when integrated. In Fig. 3.14, we see the trajectory errors and the conservation of the Hamiltonian over time given an initial condition. For integration and Poincaré plots, see Appendix Fig. 20 and Appendix Fig. 21 respectively. We again confirm that the Hamiltonian of the system can be accurately preserved using the sampled networks. Additionally, we include the integration and Poincaré plots with the same settings as in Fig. 3.14 but with $\alpha = 0.7$ in Appendix Fig. 22 and Appendix Fig. 23 respectively.

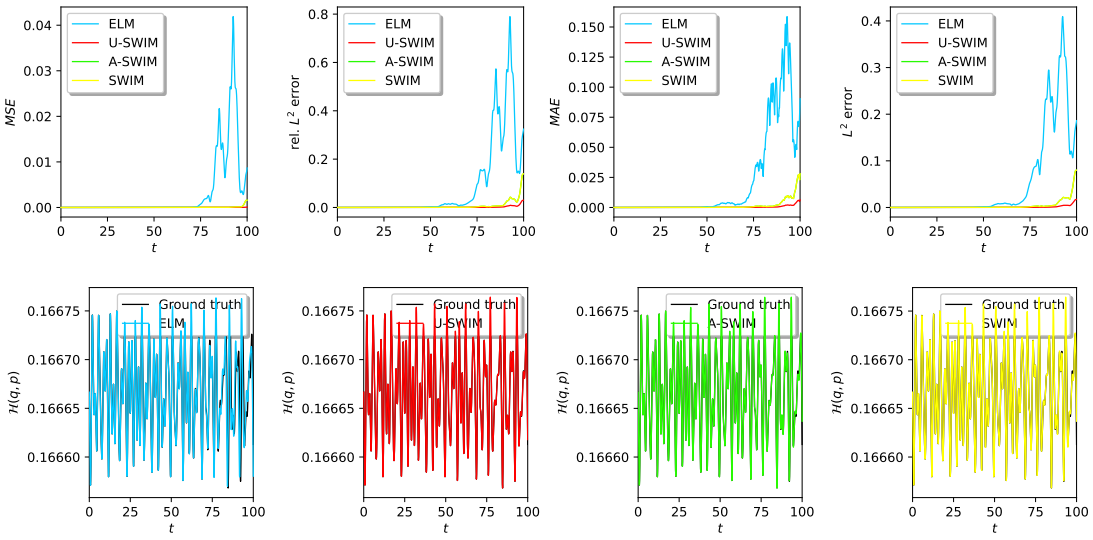


Figure 3.14: For Hénon-Heiles in Eq. (3.19) with $\alpha = 1$ trajectory errors and Hamiltonian values are plotted in the top and bottom row respectively, integrated with $h = 10^{-3}$. The median model is used (with the median approximation error on the rel. L^2 error), with 10000 neurons trained with $|\mathcal{D}| = 20000$. The Initial value is $(0, 0, 1/\sqrt{6}, 1/\sqrt{6})$. MSE is the mean squared error, and MAE is the mean absolute error. The ground truth in the bottom row represents the symplectic Euler using the true Hamiltonian.

3.3 Learning from Trajectory Data with Error Correction

Here, we demonstrate how R-HNN can learn a Hamiltonian from trajectory data consisting of position q and momentum p only without the information on the gradients or vector fields, building upon the work related to error analysis and correction explained in section 2.2.3. We assume a (trajectory) dataset is given as $\mathcal{D} = \{x_i, \varphi_h(x_i)\}_{i=1}^K$ using the exact flow φ_h of the Hamiltonian system. Note that any trajectory of arbitrary length can be rewritten to have the form of $\mathcal{D}$ if the points in the trajectory are considered pairs. Using the idea of modifying the linear system (as done in [79]), we modify the linear system we have aimed to solve in Eq. (3.4) to use finite differences instead:

$$\underbrace{\begin{bmatrix} \nabla\Phi^{(L)}(x_1) & 0 \\ \nabla\Phi^{(L)}(x_2) & 0 \\ \vdots & \vdots \\ \nabla\Phi^{(L)}(x_K) & 0 \\ \Phi^{(L)}(x_0) & 1 \end{bmatrix}}_{A\in\mathbb{R}^{(2dK+1)\times(N_L+1)}} \cdot \underbrace{\begin{bmatrix} W_{L+1}^T \\ b_{L+1} \end{bmatrix}}_{w\in\mathbb{R}^{(N_L+1)}} \overset{!}{=} \underbrace{\begin{bmatrix} \mathcal{J}^{-1}(\varphi_h(x_1) - x_1)/h \\ \mathcal{J}^{-1}(\varphi_h(x_2) - x_2)/h \\ \vdots \\ \mathcal{J}^{-1}(\varphi_h(x_K) - x_K)/h \\ \mathcal{H}(x_0) \end{bmatrix}}_{u\in\mathbb{R}^{(2dK+1)}}, \tag{3.22}$$

where we write x_i for (q_i, p_i), and use the finite differences to rewrite $\dot{x}_i$ as $(\varphi_h(x_i) - x_i)/h$. Note again that we approximate the gradient $\nabla\mathcal{H}(x)$ for each data point $x \in \mathcal{D}$ on the LHS of Eq. (3.22). Using Eq. (3.22), we give rise to the forward Euler method in the training as

$$\begin{aligned} \nabla\widehat{\mathcal{H}}(x) &\overset{!}{=} \nabla\mathcal{H}(x) \iff \\ \nabla\widehat{\mathcal{H}}(x) &\overset{\text{Eq. (2.4)}}{=} \mathcal{J}^{-1}\dot{x} \iff \\ \nabla\widehat{\mathcal{H}}(x) &\overset{\textbf{FD}}{=} \mathcal{J}^{-1}(\varphi_h(x) - x)/h \iff \\ h\mathcal{J}\nabla\widehat{\mathcal{H}}(x) &= \varphi_h(x) - x \iff \\ x + h\mathcal{J}\nabla\widehat{\mathcal{H}}(x) &= \varphi_h(x) \overset{\text{Eq. (2.4)}}{\iff} \\ x + h\widehat{\dot{x}} &= \varphi_h(x) \end{aligned} \tag{3.23}$$

for any $x \in \mathcal{D}$, where Eq. (2.4) is the Hamilton's equations, **FD** stands for finite differences, and we use the notation $\widehat{(\cdot)}$ for the terms $(\cdot)$ that are approximated. You can see forward Euler in Eq. (2.5) for validation. Similarly, we construct the linear system

$$\underbrace{\begin{bmatrix} \nabla\Phi^{(L)}(\varphi_h(q_1),p_1)) & 0 \\ \nabla\Phi^{(L)}(\varphi_h(q_2),p_2) & 0 \\ \vdots & \vdots \\ \nabla\Phi^{(L)}(\varphi_h(q_K),p_K) & 0 \\ \Phi^{(L)}(x_0) & 1 \end{bmatrix}}_{A\in\mathbb{R}^{(2dK+1)\times(N_L+1)}} \cdot \underbrace{\begin{bmatrix} W_{L+1}^T \\ b_{L+1} \end{bmatrix}}_{w\in\mathbb{R}^{(N_L+1)}} \overset{!}{=} \underbrace{\begin{bmatrix} \mathcal{J}^{-1}(\varphi_h(x_1)-x_1)/h \\ \mathcal{J}^{-1}(\varphi_h(x_2)-x_2)/h \\ \vdots \\ \mathcal{J}^{-1}(\varphi_h(x_K)-x_K)/h \\ \mathcal{H}(x_0) \end{bmatrix}}_{u\in\mathbb{R}^{(2dK+1)}}, \tag{3.24}$$

for the symplectic Euler method. For validation, you can see Eq. (2.6) and derive the symplectic Euler analogous to Eq. (3.23).

Remark. $\nabla\widehat{\mathcal{H}}(x) \overset{!}{=} \nabla\mathcal{H}(x)$ is for forward Euler, $\nabla\widehat{\mathcal{H}}(\varphi_h(q),p) \overset{!}{=} \nabla\mathcal{H}(x)$ is for symplectic Euler.

For the symplectic Euler, the learned Hamiltonian reads (taken from chapter 9, example 3.4 of [43])

$$\widehat{\mathcal{H}}(x) = \mathcal{H}(x) - \frac{h}{2}(\nabla_q\mathcal{H}(x))^T\nabla_p\mathcal{H}(x) + \mathcal{O}(h^2). \tag{3.25}$$

Post-training error correction has been studied in [21] to correct the learned Hamiltonian $\widehat{\mathcal{H}}(x)$ in Eq. (3.25) by incorporating a correction as

$$\mathcal{H}(x) \approx \widehat{\mathcal{H}}(x) + \frac{h}{2}(\nabla_q\widehat{\mathcal{H}}(x))^T\nabla_p\widehat{\mathcal{H}}(x). \tag{3.26}$$

The post-training error correction has been studied with traditional NNs; here, we experiment with it using the sampled networks (R-HNNs) without relying on automatic differentiation, as in sampled networks we can easily implement the gradient computation as explained in the previous section. Note that the correction can also be used to correct up to higher orders of h using extensions with higher order gradients.

For the single pendulum system, to approximate the Hamiltonian in Eq. (3.10), we use the symplectic Euler method and solve the linear Eq. (3.24), and plot the corrected (using Eq. (3.26)) and uncorrected rel. L^2 errors (mean of 10 runs) in Fig. 3.15. Note that we have plotted the errors of the A-SWIM method for demonstration and do not compare the R-HNN methods here.

Fig. 3.15 shows that we can reduce the error order using the sampled networks, too, and confirm the results in [21], which note similar results using the traditional NNs.

Note that one property of the learned Hamiltonian in Eq. (3.25) is that integrating it using the same integrator and time step h used in training would result in the true flow map of the system [43]. The flow map has been studied in [123, 21] for traditional NNs and in [79] for GP with error correction. As noted in [123], the networks are

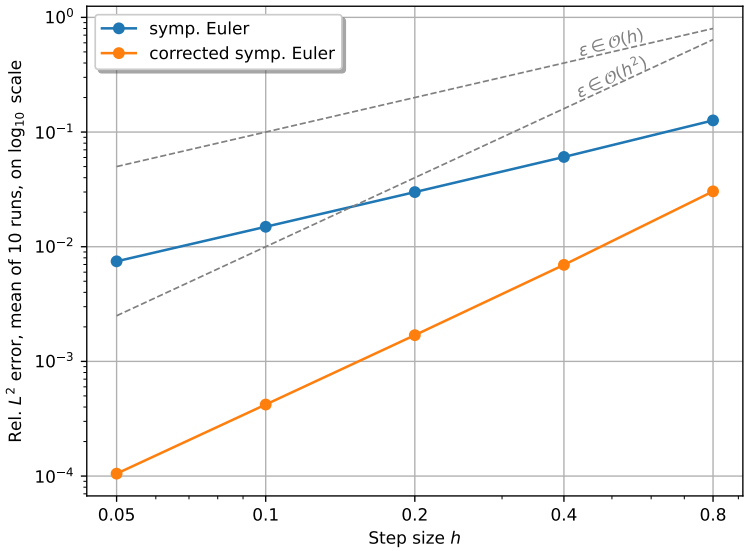


Figure 3.15: Mean Hamiltonian errors (mean of 10 runs) with and without post-training error correction are plotted on the test set of size 10000 with reference lines $\varepsilon = h$ and $\varepsilon = h^2$ using different time steps in the train set. The symplectic Euler scheme in Eq. (3.24) is solved with 200 neurons in the hidden layer and 2500 trajectories in $\mathcal{D}$ (2500 points together with their next state after time step h, which is simulated using the exact flow, which we again simulate using a very small time step 10^{-4} using symplectic Euler) within domain $[-\pi, \pi] \times [-1, 1]$.

learning the network targets rather than the Hamiltonian of the original system and can perform better than the original Hamiltonian as the network learns to compensate for the numerical errors made by the integrator as confirmed in the practical results in [18], but at the same time, reducing (or, trivially of course, increasing) the time step h used in integration in testing may potentially worsen the accuracy, as also stated in the third Remark in Proposition 1 of [21]. We do not aim to approximate the true flow map in this work. However, the modified framework in Eq. (3.24) can be used for flow map approximation experiments. We leave this for potential future work.

4 Conclusion

We present a framework for approximating Hamiltonian functions using sampled neural networks, given limited data. We evaluate and compare data-agnostic and data-drive sampling schemes and show the advantages of data-driven sampling using the SWIM algorithm in various scenarios, e.g., where the gradients of the function get large. Also, the data-agnostic ELM method needs bias configurations beforehand, which is unnecessary in SWIM-sampled networks. ELM and U-SWIM methods plateau quickly as the training set size increases, whereas SWIM and A-SWIM can provide more accurate approximations as the train set grows. However, ELM can achieve lower approximation errors with increasing network width if the bias is configured properly.

To use the SWIM algorithm with the data probabilities given limited data, we provide an initial guess as mentioned in [10] and could construct the A-SWIM method, which can match the performance of the SWIM method. Furthermore, we demonstrate that the error correction technique developed for gradient-descent-based neural networks approximating Hamiltonians can also be used for the sampled networks. This opens interesting opportunities where the performance of the sampled networks can be compared against each other, as well as the traditional neural networks, where the data is further limited, i.e., they do not contain any gradient or vector field. More systems can be tested and studied with possibly higher-order error corrections in future work; as our framework does not rely on automatic differentiation, one may more efficiently handle the gradient computations.

It would be beneficial to compare the Hamiltonian approximation performance of traditional neural networks and sampled neural networks. Furthermore, this work assumes symplecticity, but in reality, most systems have dissipation that needs to be accounted for. Also, the real-life data can be noisy, which is another important consideration.

In addition, using sampled networks to learn the flow map of Hamiltonian systems can lead to faster runtimes. This can help to compensate for the time lost when back-propagating through an integrator. For example, the longer training times reported in [116] using well-studied higher-order integrators may be reduced by utilizing sampled networks. Also, the linear system in Eq. (3.24) can be studied for learning the exact flow, as already done with Gaussian Processes in [79].

Bibliography

[1] C. Allen-Blanchette. "Hamiltonian GAN." In: *arXiv preprint arXiv:2308.11216* (2023).

[2] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Third. Society for Industrial and Applied Mathematics, 1999.

[3] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, et al. "Learning dexterous in-hand manipulation." In: *The International Journal of Robotics Research* 39.1 (2020), pp. 3–20.

[4] G. Arulampalam and A. Bouzerdoum. "A generalized feedforward neural network architecture for classification and regression." In: *Neural networks* 16.5-6 (2003), pp. 561–568.

[5] F. de Avila Belbute-Peres, K. Smith, K. Allen, J. Tenenbaum, and J. Z. Kolter. "End-to-end differentiable physics for learning and control." In: *Advances in neural information processing systems* 31 (2018).

[6] J. Baxter. "A model of inductive bias learning." In: *Journal of artificial intelligence research* 12 (2000), pp. 149–198.

[7] T. Bertalan, F. Dietrich, I. Mezí, and I. G. Kevrekidis. "On learning Hamiltonian systems from data." In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 29.12 (2019). DOI: 10.1063/1.5128231.

[8] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[9] J. Blechschmidt and O. G. Ernst. "Three ways to solve partial differential equations with neural networks—A review." In: *GAMM-Mitteilungen* 44.2 (2021), e202100006.

[10] E. L. Bolager, I. Burak, C. Datar, Q. Sun, and F. Dietrich. *Sampling weights of deep neural networks*. 2023. arXiv: 2306.16830 [cs.LG].

[11] F. Brezzi and M. Fortin. *Mixed and hybrid finite element methods*. Vol. 15. Springer Science & Business Media, 2012.

[12] S. L. Brunton and J. N. Kutz. *Machine Learning for Partial Differential Equations*. 2023. arXiv: 2303.17078 [cs.LG].

[13] M. B. Chang, T. Ullman, A. Torralba, and J. B. Tenenbaum. "A compositional object-based approach to learning physical dynamics." In: *arXiv* (2016).

[14] Y. Chauvin and D. E. Rumelhart. *Backpropagation: theory, architectures, and applications*. Psychology press, 2013.

[15] J. Chen, X. Chi, Z. Yang, et al. "Bridging traditional and machine learning-based algorithms for solving PDEs: the random feature method." In: *J Mach Learn* 1 (2022), pp. 268–98.

[16] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. "Neural ordinary differential equations." In: *Advances in neural information processing systems* 31 (2018).

[17] Y. Chen, T. Matsubara, and T. Yaguchi. "Neural Symplectic Form: Learning Hamiltonian Equations on General Coordinate Systems." In: *Advances in Neural Information Processing Systems*. 2021.

[18] Z. Chen, J. Zhang, M. Arjovsky, and L. Bottou. "Symplectic recurrent neural networks." In: *arXiv preprint arXiv:1909.13334* (2019).

[19] K. Course, T. Evans, and P. Nair. "Weak form generalized hamiltonian learning." In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 18716–18726.

[20] M. Cranmer, S. Greydanus, S. Hoyer, P. Battaglia, D. Spergel, and S. Ho. "Lagrangian neural networks." In: *arXiv preprint arXiv:2003.04630* (2020).

[21] M. David and F. Méhats. "Symplectic learning for Hamiltonian neural networks." In: *Journal of Computational Physics* 494 (2023), p. 112495.

[22] R. De Vogelaere. "Methods of integration which preserve the contact transformation property of the Hamilton equations." In: *Technical report (University of Notre Dame. Dept. of Mathematics)* (1956).

[23] S. A. Desai, M. Mattheakis, D. Sondak, P. Protopapas, and S. J. Roberts. "Port-Hamiltonian neural networks for learning explicit time-dependent dynamical systems." In: *Physical Review E* 104.3 (2021), p. 034312.

[24] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding." In: *arXiv* (2018).

[25] D. DiPietro, S. Xiong, and B. Zhu. "Sparse symplectically integrated neural networks." In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 6074–6085.

[26] T. P. Duong and N. Atanasov. "Hamiltonian-based Neural ODE Networks on the SE(3) Manifold For Dynamics Learning and Control." In: (2021).

[27] K. Duru, L. Rannabauer, A.-A. Gabriel, G. Kreiss, and M. Bader. "A stable discontinuous Galerkin method for the perfectly matched layer for elastodynamics in first order form." In: *Numerische Mathematik* 146 (2020), pp. 729–782.

[28] *Elements of Applied Bifurcation Theory*. 2004.

[29] A. Erdemir, T. M. Guess, J. Halloran, S. C. Tadepalli, and T. M. Morrison. "Considerations for reporting finite element analysis studies in biomechanics." In: *Journal of biomechanics* 45.4 (2012), pp. 625–633.

[30] K. Feng. "On difference schemes and symplectic geometry." In: *Proceedings of the 5th international symposium on differential geometry and differential equations*. 1984.

[31] R. L. Fernandes and W. M. Oliva. "Hamiltonian dynamics of the Lotka-Volterra equations." In: *International Conference on Differential Equations, Lisboa*. World Scientific. 1995, pp. 327–334.

[32] E. Ferrer, G. Rubio, G. Ntoukas, W. Laskowski, O. Mariño, S. Colombo, A. Mateo-Gabın, H. Marbona, F. M. de Lara, D. Huergo, et al. ": A high-order discontinuous Galerkin solver for flow simulations and multi-physics applications." In: *Computer Physics Communications* 287 (2023), p. 108700.

[33] E. Forest and R. D. Ruth. "Fourth-order symplectic integration." In: *Physica D: Nonlinear Phenomena* 43.1 (1990), pp. 105–117. ISSN: 0167-2789. DOI: https://doi.org/10.1016/0167-2789(90)90019-L.

[34] N. Galioto and A. A. Gorodetsky. "Bayesian identification of Hamiltonian dynamics from symplectic data." In: *2020 59th IEEE Conference on Decision and Control (CDC)*. IEEE. 2020, pp. 1190–1195.

[35] C. Gallicchio and S. Scardapane. "Deep randomized neural networks." In: *Recent Trends in Learning From Data: Tutorials from the INNS Big Data and Deep Learning Conference (INNSBDDL2019)*. Springer. 2020, pp. 43–68.

[36] L. Gonon. "Random feature neural networks learn Black-Scholes type PDEs without curse of dimensionality." In: *Journal of Machine Learning Research* 24.189 (2023), pp. 1–51.

[37] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[38] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. "Generative adversarial nets." In: *Advances in neural information processing systems* 27 (2014).

[39] M. Gori, A. Tesi, et al. "On the problem of local minima in backpropagation." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.1 (1992), pp. 76–86.

[40] S. Greydanus, M. Dzamba, and J. Yosinski. "Hamiltonian neural networks." In: *Advances in neural information processing systems* 32 (2019).

[41] M. Gu, J. Demmel, and I. Dhillon. *Efficient computation of the singular value decomposition with applications to least squares problems*. Computer Science Department, 1994.

[42] E. Hairer. "Challenges in geometric numerical integration." In: *Trends in contemporary mathematics* (2014), pp. 125–135.

[43] E. Hairer, M. Hochbruck, A. Iserles, and C. Lubich. "Geometric numerical integration." In: *Oberwolfach Reports* 3.1 (2006), pp. 805–882.

[44] R. Hamid, Y. Xiao, A. Gittens, and D. DeCoste. "Compact random feature maps." In: *International conference on machine learning*. PMLR. 2014, pp. 19–27.

[45] W. R. Hamilton. "On a General Method in Dynamics." In: *Philosophical Transactions of the Royal Society* 124 (1834), pp. 247–308.

[46] W. R. Hamilton. "Second Essay on a General Method in Dynamics." In: *Philosophical Transactions of the Royal Society* 125 (1835), pp. 95–144.

[47] C.-D. Han, B. Glaz, M. Haile, and Y.-C. Lai. "Adaptable Hamiltonian neural networks." In: *Physical Review Research* 3.2 (2021), p. 023156.

[48] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. "Array programming with NumPy." In: *Nature* 585.7825 (2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2.

[49] D. Haussler. "Quantifying inductive bias: AI learning algorithms and Valiant's learning framework." In: *Artificial intelligence* 36.2 (1988), pp. 177–221.

[50] K. He, X. Zhang, S. Ren, and J. Sun. "Deep residual learning for image recognition." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[51] M. Hénon and C. Heiles. "The applicability of the third integral of motion: some numerical experiments." In: *Astronomical Journal, Vol. 69, p. 73 (1964)* 69 (1964), p. 73.

[52] P. Horn, V. Ulibarrena, B. Koren, and S. Zwart. "Structure-preserving neural networks for the n-body problem." In: *8th European Congress on Computational Methods in Applied Sciences and Engineering, ECCOMAS Congress 2022*. 2022. DOI: 10.23967/eccomas.2022.262.

[53] K. Hornik, M. Stinchcombe, and H. White. "Multilayer feedforward networks are universal approximators." In: *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8.

[54] H. H. Hu. "Computational fluid dynamics." In: *Fluid mechanics*. Elsevier, 2012, pp. 421–472.

[55] G.-B. Huang, L. Chen, and C. Siew. "Universal Approximation Using Incremental Constructive Feedforward Networks With Random Hidden Nodes." In: *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council* 17 (2006), pp. 879–92. DOI: 10.1109/TNN.2006.875977.

[56] G.-B. Huang, Q.-Y. Zhu, and C. Siew. "Extreme learning machine: A new learning scheme of feedforward neural networks." In: vol. 2. 2004, 985–990 vol.2. ISBN: 0-7803-8359-1. DOI: 10.1109/IJCNN.2004.1380068.

[57] R. Iten, T. Metger, H. Wilming, L. Del Rio, and R. Renner. "Discovering physical concepts with neural networks." In: *Physical review letters* 124.1 (2020), p. 010508.

[58] R. A. Jacobs. "Increased rates of convergence through learning rate adaptation." In: *Neural networks* 1.4 (1988), pp. 295–307.

[59] H. Jaeger. "The "echo state" approach to analysing and training recurrent neural networks-with an erratum note." In: *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report* 148.34 (2001), p. 13.

[60] A. Jakovác, M. T. Kurbucz, and P. Pósfay. "Reconstruction of observed mechanical motions with artificial intelligence tools." In: *New Journal of Physics* 24.7 (2022), p. 073021. ISSN: 1367-2630. DOI: 10.1088/1367-2630/ac7c2d.

[61] P. Jin, Z. Zhang, I. G. Kevrekidis, and G. E. Karniadakis. "Learning Poisson systems and trajectories of autonomous systems via Poisson neural networks." In: *IEEE Transactions on Neural Networks and Learning Systems* (2022).

[62] P. Kar and H. Karnick. "Random feature maps for dot product kernels." In: *Artificial intelligence and statistics*. PMLR. 2012, pp. 583–591.

[63] Z.-Y. Khoo, J. S. C. Low, and S. Bressan. "Separable Hamiltonian Neural Networks." In: *arXiv preprint arXiv:2309.01069* (2023).

[64] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira, C. Burges, L. Bottou, and K. Weinberger. Vol. 25. Curran Associates, Inc., 2012.

[65] S. Larsson and V. Thomée. *Partial differential equations with numerical methods*. Vol. 45. Springer, 2003.

[66] Q. Le, T. Sarlós, and A. Smola. "Fastfood-computing hilbert space expansions in loglinear time." In: *International Conference on Machine Learning*. PMLR. 2013, pp. 244–252.

[67] B. Leimkuhler and S. Reich. *Simulating hamiltonian dynamics*. 14. Cambridge university press, 2004.

[68] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken. "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function." In: *Neural networks* 6.6 (1993), pp. 861–867.

[69] H. C. Leung, C. S. Leung, and E. W. M. Wong. "Fault and Noise Tolerance in the Incremental Extreme Learning Machine." In: *IEEE Access* 7 (2019), pp. 155171–155183.

[70] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen. "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection." In: *The International journal of robotics research* 37.4-5 (2018), pp. 421–436.

[71] S. Liang, Z. Huang, and H. Zhang. "Stiffness-aware neural network for learning Hamiltonian systems." In: *International Conference on Learning Representations*. 2021.

[72] M. Lukoševičius and H. Jaeger. "Reservoir computing approaches to recurrent neural network training." In: *Computer science review* 3.3 (2009), pp. 127–149.

[73] G. D. Magoulas, M. N. Vrahatis, and G. S. Androulakis. "Improving the convergence of the backpropagation algorithm using learning rate adaptation methods." In: *Neural Computation* 11.7 (1999), pp. 1769–1796.

[74] J. S. Manoharan. "Study of variants of extreme learning machine (ELM) brands and its performance measure on classification algorithm." In: *Journal of Soft Computing Paradigm (JSCP)* 3.02 (2021), pp. 83–95.

[75] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015.

[76] T. Matsubara, A. Ishikawa, and T. Yaguchi. "Deep energy-based modeling of discrete-time physics." In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 13100–13111.

[77] M. Mattheakis, D. Sondak, A. S. Dogra, and P. Protopapas. "Hamiltonian neural networks for solving equations of motion." In: *Physical Review E* 105.6 (2022), p. 065305.

[78] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. "Playing Atari with Deep Reinforcement Learning." In: abs/1312.5602 (2013). arXiv: 1312.5602.

[79] C. Offen and S. Ober-Blöbaum. "Symplectic integration of learned Hamiltonian systems." In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 32.1 (2022).

[80] G. Pang and G. E. Karniadakis. "Physics-Informed Learning Machines for Partial Differential Equations: Gaussian Processes Versus Neural Networks." In: *Emerging Frontiers in Nonlinear Science*. Ed. by P. G. Kevrekidis, J. Cuevas-Maraver, and A. Saxena. Cham: Springer International Publishing, 2020, pp. 323–343.

[81] Y.-H. Pao and Y. Takefuji. "Functional-link net computing: theory, system architecture, and functionalities." In: *Computer* 25.5 (1992), pp. 76–79.

[82] Y.-H. Pao, G.-H. Park, and D. J. Sobajic. "Learning and generalization characteristics of the random vector functional-link net." In: *Neurocomputing* 6.2 (1994), pp. 163–180.

[83] J. Park and I. W. Sandberg. "Universal approximation using radial-basis-function networks." In: *Neural computation* 3.2 (1991), pp. 246–257.

[84] A. Rahimi and B. Recht. "Random features for large-scale kernel machines." In: *Advances in neural information processing systems* 20 (2007).

[85] A. Rahimi and B. Recht. "Uniform approximation of functions with random bases." In: *2008 46th annual allerton conference on communication, control, and computing*. IEEE. 2008, pp. 555–561.

[86] 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." In: *Journal of Computational Physics* 378 (2019), pp. 686–707.

[87] L. Rannabauer, M. Dumbser, and M. Bader. "ADER-DG with a-posteriori finite-volume limiting to simulate tsunamis in a parallel adaptive mesh refinement framework." In: *Computers & Fluids* 173 (2018), pp. 299–306.

[88] C. E. Rasmussen. "Gaussian Processes in Machine Learning." In: *Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2 - 14, 2003, Tübingen, Germany, August 4 - 16, 2003, Revised Lectures*. Springer, 2004, pp. 63–71. ISBN: 978-3-540-28650-9. DOI: 10.1007/978-3-540-28650-9_4.

[89] L. E. Reichl and J. H. Luscombe. *A Modern Course in Statistical Physics, 2nd Edition*. Vol. 67. 12. 1999, pp. 1285–1287.

[90] A. Reinarz, D. E. Charrier, M. Bader, L. Bovard, M. Dumbser, K. Duru, F. Fambri, A.-A. Gabriel, J.-M. Gallard, S. Köppel, et al. "ExaHyPE: An engine for parallel dynamically adaptive simulations of wave problems." In: *Computer Physics Communications* 254 (2020), p. 107251.

[91] J. J. Sakurai, S. F. Tuan, and E. D. Commins. *Modern Quantum Mechanics, Revised Edition*. Vol. 63. 1. 1995, pp. 93–95.

[92] A. Sanchez-Gonzalez, V. Bapst, K. Cranmer, and P. Battaglia. "Hamiltonian graph networks with ode integrators." In: *arXiv preprint arXiv:1909.12790* (2019).

[93] I. H. Sarker. "Machine learning: Algorithms, real-world applications and research directions." In: *SN computer science* 2.3 (2021), p. 160.

[94] V. Saz Ulibarrena, P. Horn, S. Portegies Zwart, E. Sellentin, B. Koren, and M. X. Cai. "A hybrid approach for solving the gravitational N-body problem with Artificial Neural Networks." In: *Journal of Computational Physics* 496 (2024), p. 112596. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2023.112596.

[95] J. Schmidhuber. "Deep learning in neural networks: An overview." In: *Neural networks* 61 (2015), pp. 85–117.

[96] M. Schmidt and H. Lipson. "Distilling free-form natural laws from experimental data." In: *science* 324.5923 (2009), pp. 81–85.

[97] W. F. Schmidt, M. A. Kraaijveld, R. P. Duin, et al. "Feed forward neural networks with random weights." In: *International conference on pattern recognition*. IEEE Computer Society Press. 1992, pp. 1–1.

[98] H. Sharma, Z. Wang, and B. Kramer. "Hamiltonian operator inference: Physics-preserving learning of reduced-order models for canonical Hamiltonian systems." In: *Physica D: Nonlinear Phenomena* 431 (2022), p. 133122.

[99] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. "Mastering the game of go without human knowledge." In: *nature* 550.7676 (2017), pp. 354–359.

[100] A. Sosanya and S. Greydanus. "Dissipative hamiltonian neural networks: Learning dissipative and conservative dynamics separately." In: *arXiv* (2022).

[101] P. N. Suganthan and R. Katuwal. "On the origins of randomization-based feedforward neural networks." In: *Applied Soft Computing* 105 (2021), p. 107239.

[102] I. Sutskever, O. Vinyals, and Q. V. Le. "Sequence to sequence learning with neural networks." In: *Advances in neural information processing systems* 27 (2014).

[103] "SympNets: Intrinsic structure-preserving symplectic networks for identifying Hamiltonian systems." In: *Neural Networks* 132 (2020), pp. 166–179. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2020.08.017.

[104] K.-A. Tan, R. Morison, and L. Leslie. "A comparison of high-order explicit and non-oscillatory finite difference advection schemes for climate and weather models." In: *Meteorology and Atmospheric Physics* 89 (2005), pp. 251–267.

[105] Y. Tanaka, T. Iwata, and n. ueda naonori. "Symplectic Spectrum Gaussian Processes: Learning Hamiltonians from Noisy and Sparse Data." In: *Advances in Neural Information Processing Systems*. Ed. by S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh. Vol. 35. Curran Associates, Inc., 2022, pp. 20795–20808.

[106] M. Tao. "Explicit symplectic approximation of nonseparable Hamiltonians: Algorithm and long time performance." In: *Physical Review E* 94.4 (2016), p. 043303.

[107] J. R. Taylor. *Classical mechanics*. Vol. 1. Springer, 2005.

[108] J. B. Tenenbaum, V. d. Silva, and J. C. Langford. "A global geometric framework for nonlinear dimensionality reduction." In: *science* 290.5500 (2000), pp. 2319–2323.

[109] V. A. Titarev and E. F. Toro. "ADER: Arbitrary high order Godunov approach." In: *Journal of Scientific Computing* 17 (2002), pp. 609–618.

[110] Y. Tong, S. Xiong, X. He, G. Pan, and B. Zhu. "Symplectic Neural Networks in Taylor Series Form for Hamiltonian Systems." In: (2021).

[111] P. Toth, D. J. Rezende, A. Jaegle, S. Racanière, A. Botev, and I. Higgins. "Hamiltonian generative networks." In: *arXiv preprint arXiv:1909.13789* (2019).

[112] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python." In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[113] J. Wang, S. Lu, S.-H. Wang, and Y.-D. Zhang. "A review on extreme learning machine." In: *1181: Multimedia-based Healthcare Systems using Computational Intelligence* 81 (2022), pp. 41611–41660. DOI: https://doi.org/10.1007/s11042-021-11007-7.

[114] N. Watters, D. Zoran, T. Weber, P. Battaglia, R. Pascanu, and A. Tacchetti. "Visual interaction networks: Learning a physics simulator from video." In: *Advances in neural information processing systems* 30 (2017).

[115] K. Wu, T. Qin, and D. Xiu. "Structure-preserving method for reconstructing unknown Hamiltonian systems from trajectory data." In: *SIAM Journal on Scientific Computing* 42.6 (2020), A3704–A3729.

[116] S. Xiong, Y. Tong, X. He, S. Yang, C. Yang, and B. Zhu. "Nonseparable Symplectic Neural Networks." In: (2022).

[117] S. Yildiz, P. Goyal, T. Bendokat, and P. Benner. *Data-Driven Identification of Quadratic Symplectic Representations of Nonlinear Hamiltonian Systems*. 2023.

[118] J. Yosinski, J. Clune, D. Hidalgo, S. Nguyen, J. C. Zagal, and H. Lipson. "Evolving robot gaits in hardware: The hyperneat generative encoding vs. parameter optimization." In: *Proceedings of the 20th European Conference on Artificial Life* (2011), pp. 890–897.

[119] L. Zhang and P. N. Suganthan. "A survey of randomized algorithms for training neural networks." In: *Information Sciences* 364 (2016), pp. 146–155.

[120] R. Zhang, Y. Lan, G.-B. Huang, and Z.-B. Xu. "Universal Approximation of Extreme Learning Machine With Adaptive Growth of Hidden Nodes." In: *IEEE Transactions on Neural Networks and Learning Systems* 23.2 (2012), pp. 365–371. DOI: 10.1109/TNNLS.2011.2178124.

[121] Y. D. Zhong, B. Dey, and A. Chakraborty. "Dissipative SymODEN: Encoding Hamiltonian Dynamics with Dissipation and Control into Deep Learning." In: *8th International Conference on Learning Representations, ICLR 2020, Workshop on Integration of Deep Neural Models and Differential Equations (DeepDiffEq)*. 2020.

[122] Y. D. Zhong, B. Dey, and A. Chakraborty. "Symplectic ODE-Net: Learning Hamiltonian Dynamics with Control." In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. 2020.

[123] A. Zhu, P. Jin, and Y. Tang. "Deep Hamiltonian networks based on symplectic integrators." In: *arXiv* (2020).

Appendix

Here are the rest of the plots mentioned in the numerical experiments section 3.2.

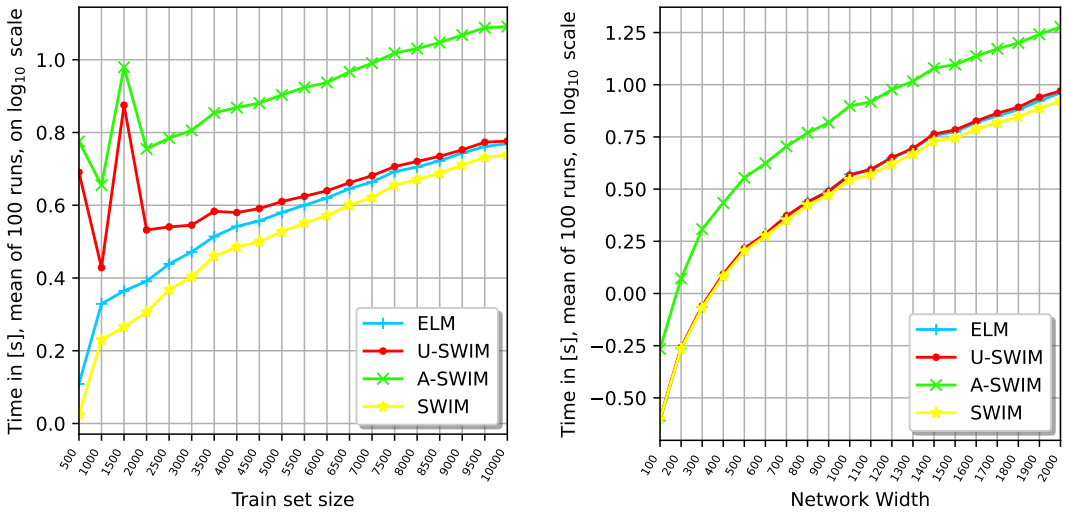


Figure 1: Mean times took for training in domain $[-2\pi, 2\pi] \times [-6, 6]$ for single pendulum system in Eq. (3.8) with $|\mathcal{D}|$ scaling with network width set to 1500 are plotted on the left, and network width scaling with $|\mathcal{D}| = 10000$ on the right. The results are analogous for domain $[-2\pi, 2\pi] \times [-1, 1]$. The spikes in the train times for the low number of train set sizes may indicate duplicate points picked for the weight sampling which needed to be re-sampled for the SWIM methods. A-SWIM train times follow the U-SWIM lines at these spikes, since in A-SWIM the initial fit is done using the U-SWIM method.

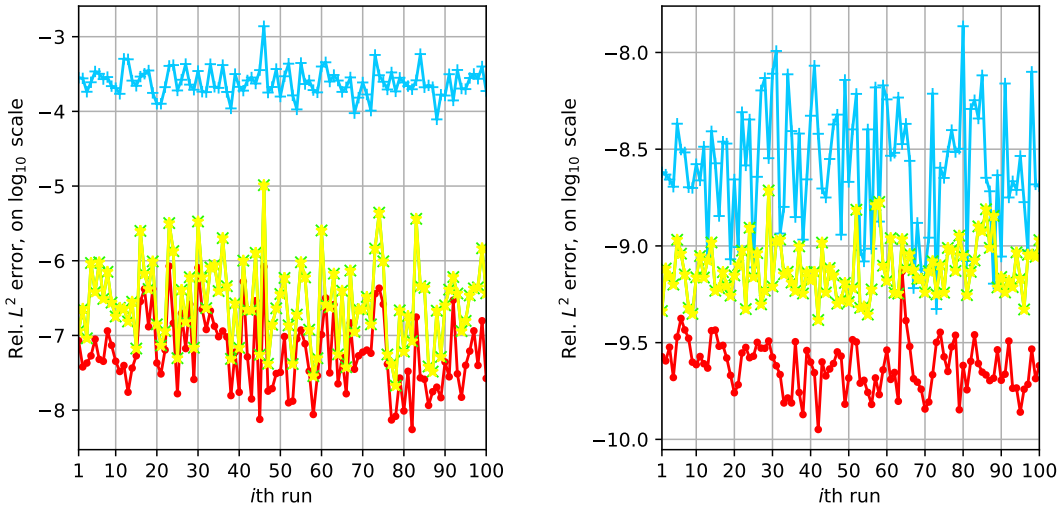


Figure 2: For single pendulum in Eq. (3.10) Hamiltonian errors on $\mathcal{Z}$ are plotted, where $|\mathcal{Z}| = 10000$. On the left, the networks are trained with $|\mathcal{D}| = 500$, and on the right, with $|\mathcal{D}| = 10000$. In both setting the networks have 1500 neurons in the hidden layer, and are trained in domain $[-2\pi, 2\pi] \times [-6, 6]$.

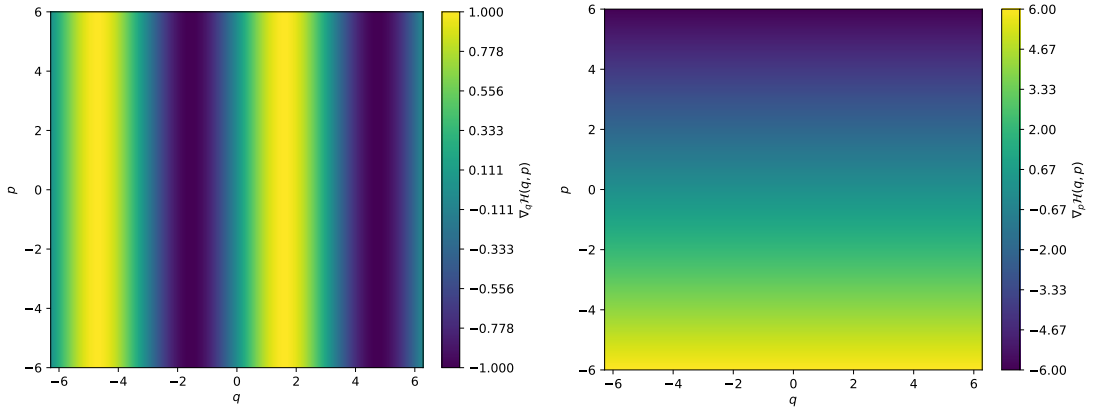


Figure 3: Gradient of the ground truth Hamiltonian for the single pendulum system in Eq. (3.10).

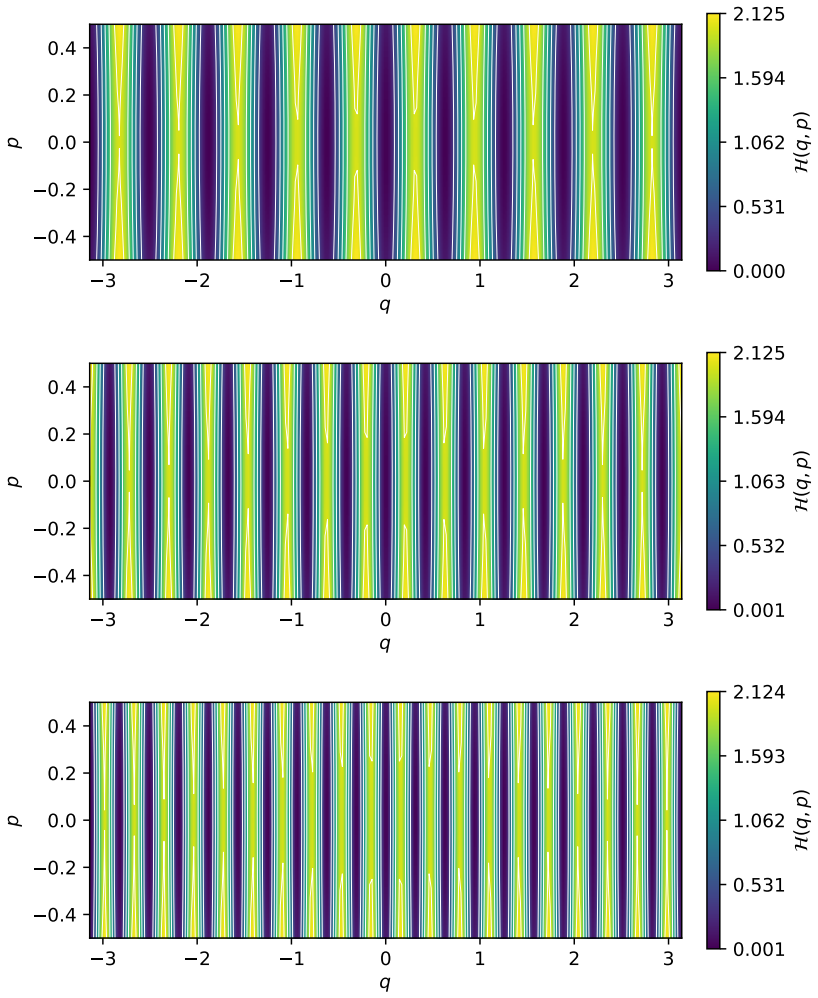


Figure 4: Ground truth phase plots of the single pendulum in Eq. (3.11) with different frequencies ($f = 10$ at the top, $f = 15$ in the middle, $f = 20$ at the bottom) for domain $[\pi, \pi] \times [-0.5, 0.5]$.

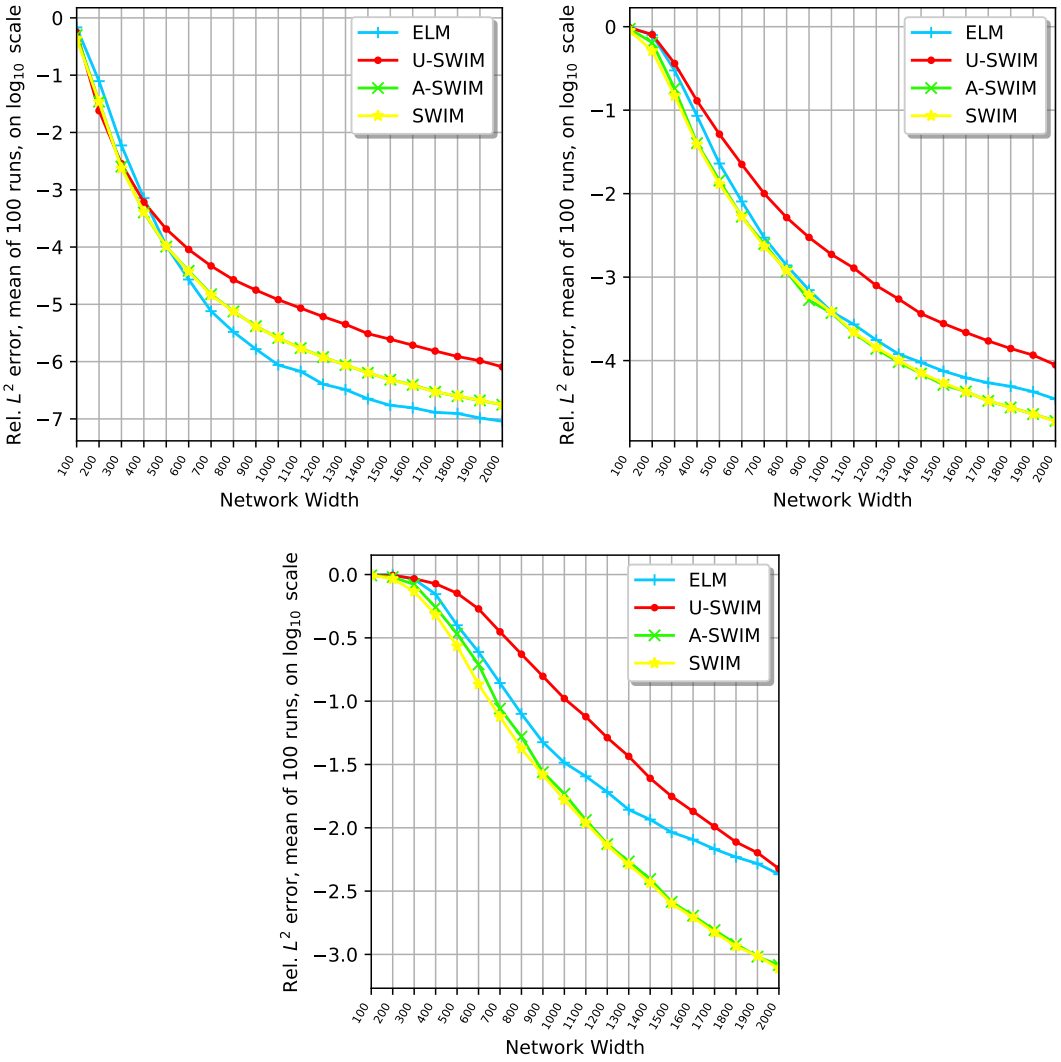


Figure 5: For single pendulum in Eq. (3.11) Hamiltonian errors on $\mathcal{Z}$ are plotted, where $|\mathcal{Z}| = 10000$. All the settings are trained in domain $[\pi, \pi] \times [-1, 1]$ with $|\mathcal{D}| = 10000$. Network width scaling is plotted for different settings. Frequency (f) is set to 10 at the top left, 15 at the top right, and 20 at the bottom.

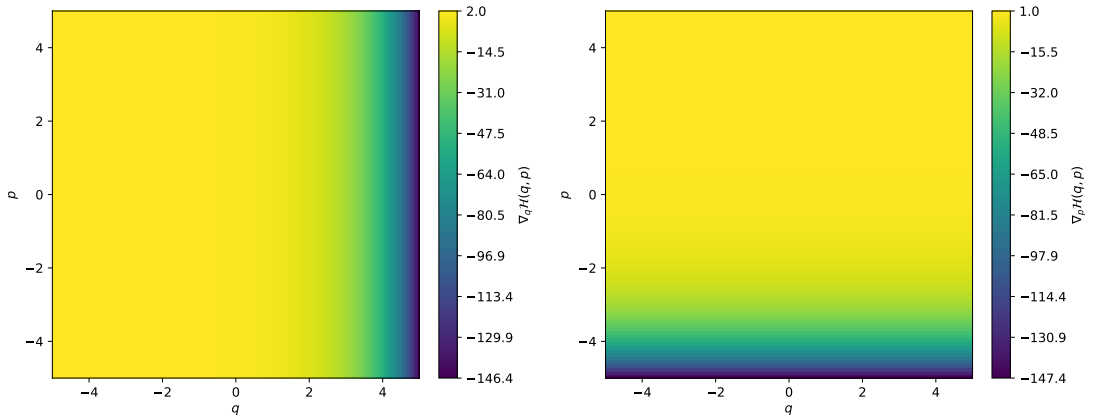


Figure 6: Gradient of the ground truth Hamiltonian for the Lotka-Volterra in Eq. (3.15).

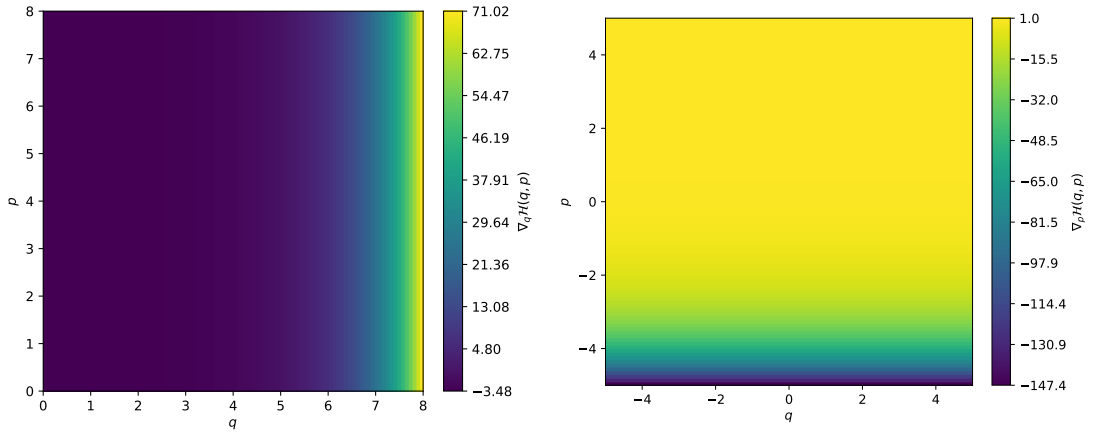


Figure 7: Gradient of the ground truth Hamiltonian for the Lotka-Volterra in Eq. (3.16).

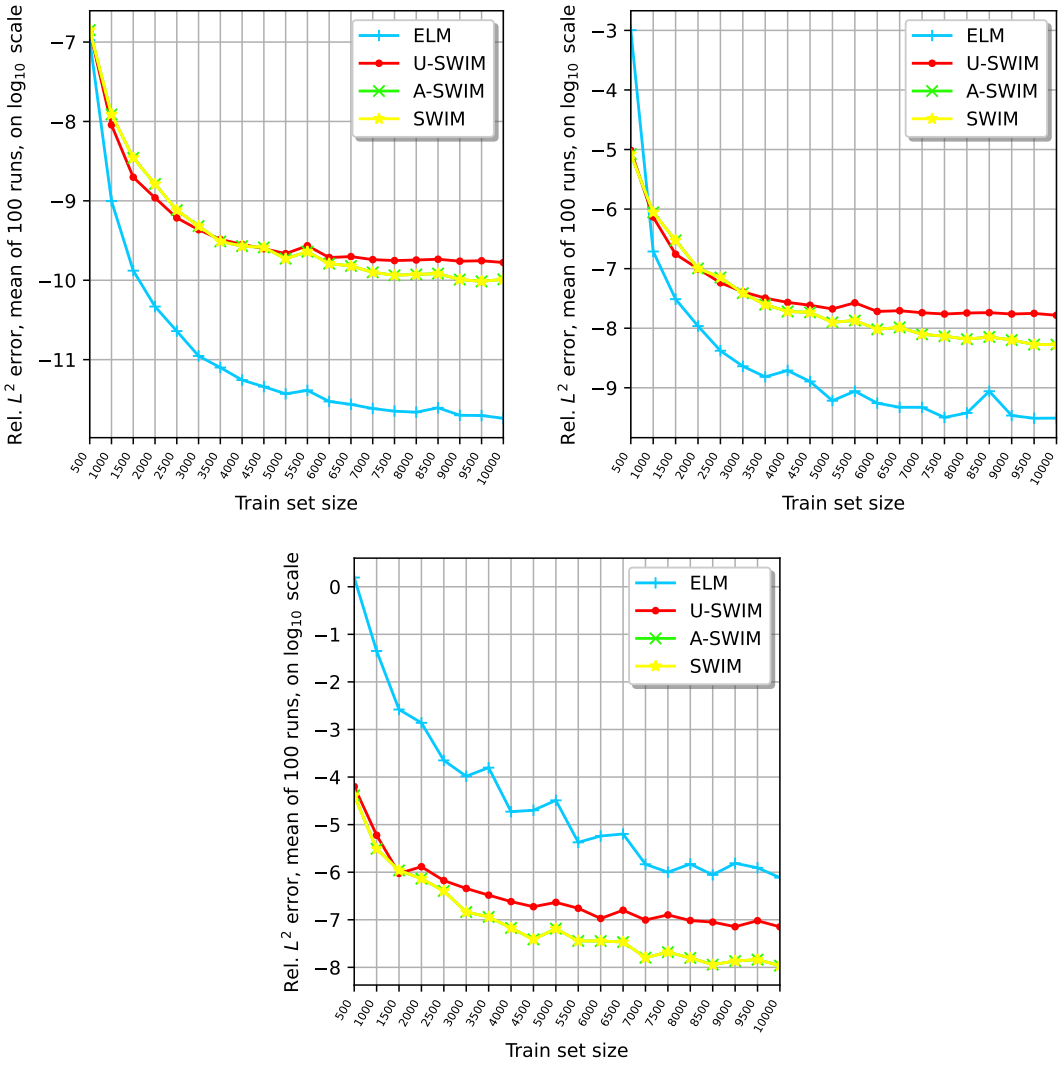


Figure 8: For Lotka-Volterra Hamiltonian errors on $\mathcal{Z}$ are plotted with $|\mathcal{D}|$ scaling. The top left and the top right plots are zero-centered, as in Eq. (3.15), and are trained in domains $[-1,1]^2$ and $[-5,5]^2$ respectively. The bottom plot is centered around $(5,5)$, as in Eq. (3.16), and is trained in domain $[0,8]^2$. All the settings are trained with 1500 network width and tested with $|\mathcal{Z}| = 10000$.

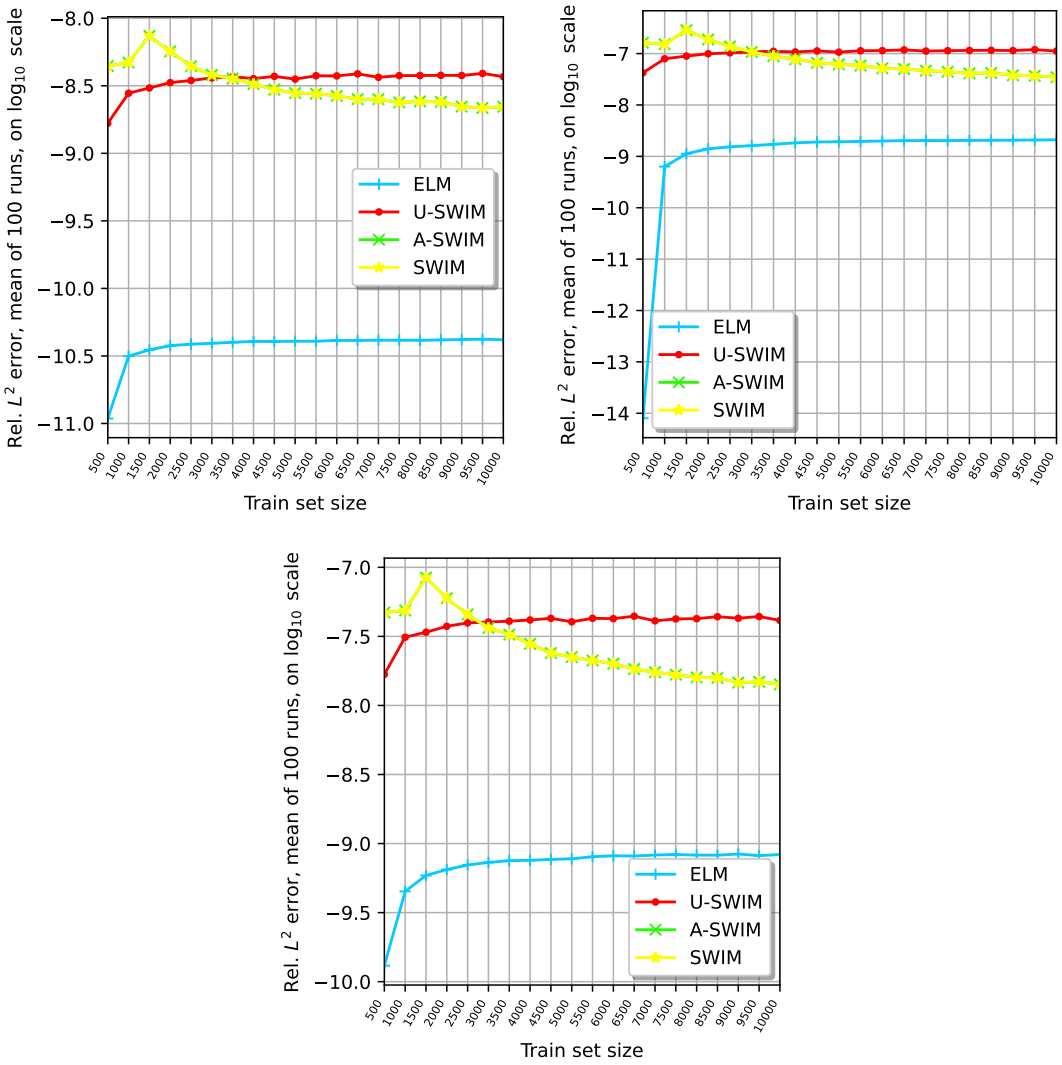


Figure 9: For Lotka-Volterra gradient errors on $\mathcal{D}$ are plotted with $|\mathcal{D}|$ scaling. The top left and the top right plots are zero-centered, as in Eq. (3.15), and are trained in domains $[-1,1]^2$ and $[-5,5]^2$ respectively. The bottom plot is centered around $(5,5)$, as in Eq. (3.16), and is trained in domain $[0,8]^2$. All the settings are trained with 1500 network width.

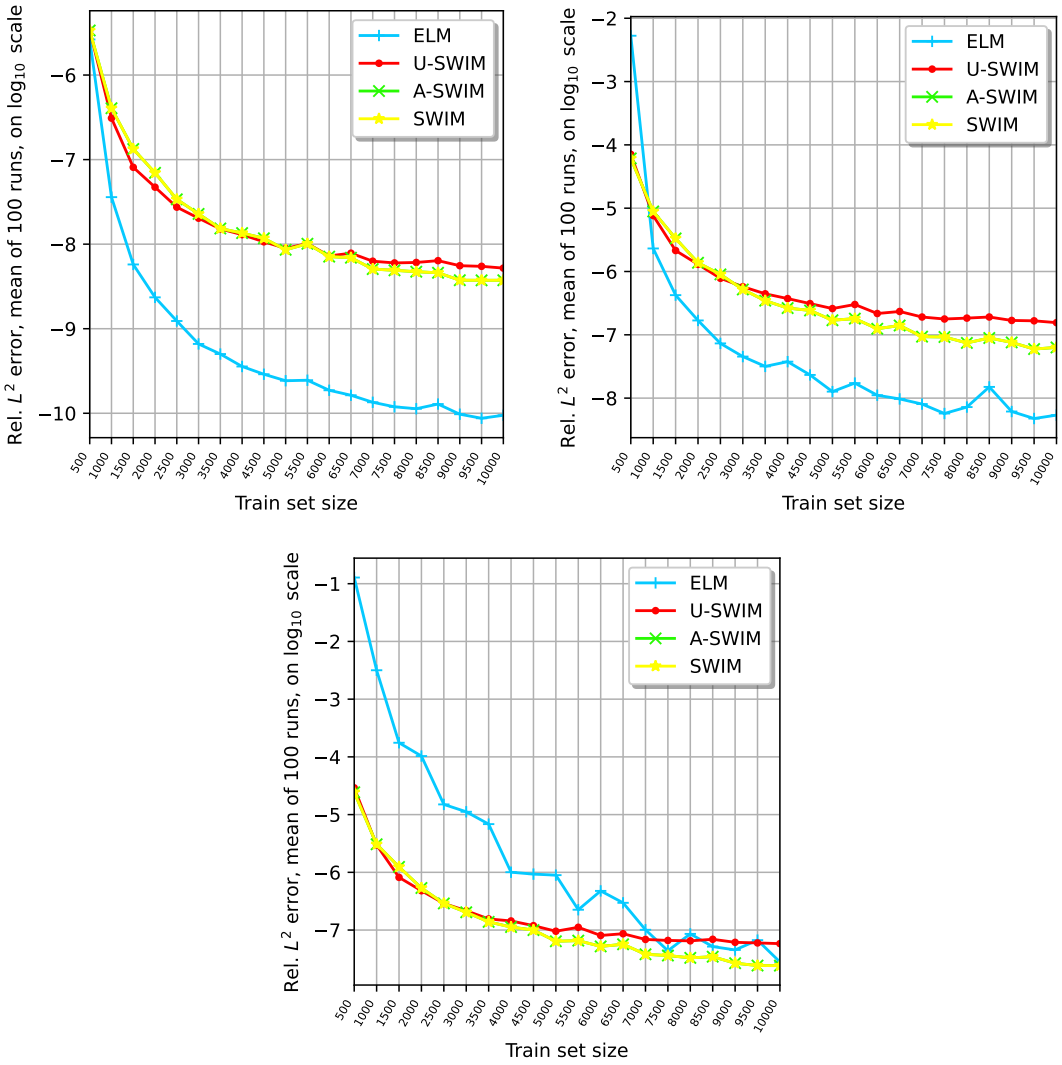


Figure 10: For Lotka-Volterra gradient errors on $\mathcal{Z}$ are plotted with $|\mathcal{D}|$ scaling. The top left and the top right plots are zero-centered, as in Eq. (3.15), and are trained in domains $[-1,1]^2$ and $[-5,5]^2$ respectively. The bottom plot is centered around $(5,5)$, as in Eq. (3.16), and is trained in domain $[0,8]^2$. All the settings are trained with 1500 network width and tested with $|\mathcal{Z}| = 10000$.

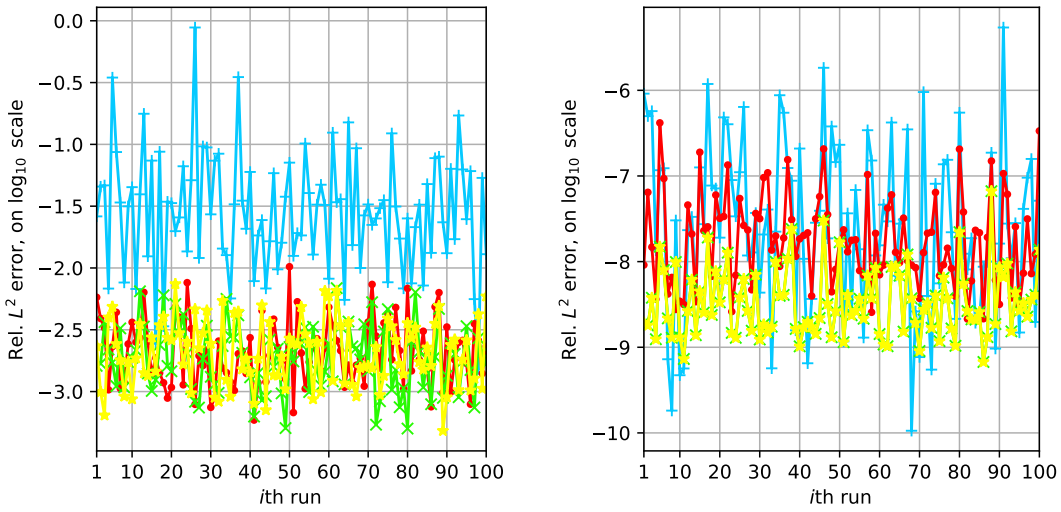


Figure 11: For Lotka-Volterra in Eq. (3.16) Hamiltonian errors on $\mathcal{Z}$ are plotted. On the left, the networks are trained with 100 neurons in the hidden layer, and on the right, 2000 neurons. In both settings, the networks are trained with $|\mathcal{D}| = 10000$ in domain $[0, 8]^2$, and tested with $|\mathcal{Z}| = 10000$.

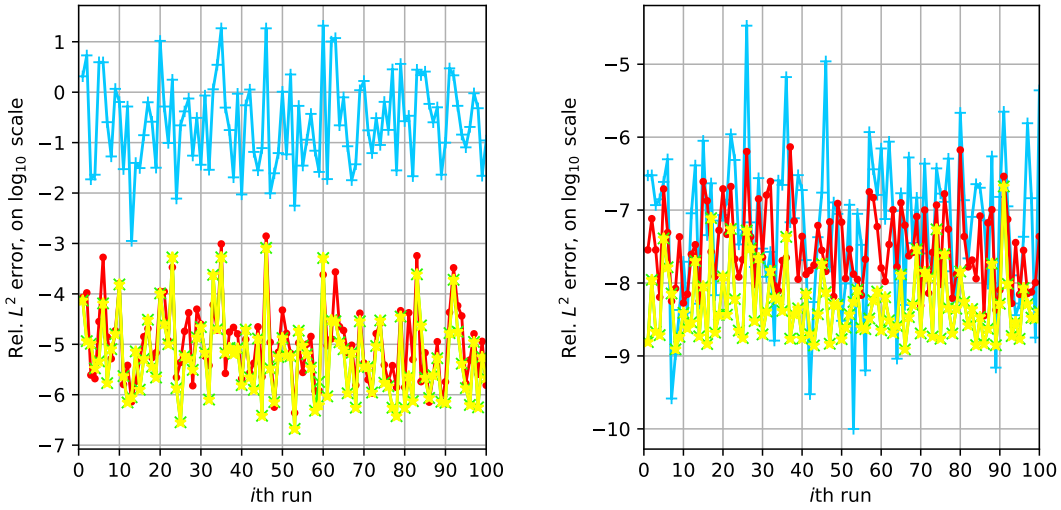


Figure 12: For Lotka-Volterra in Eq. (3.16) Hamiltonian errors on $\mathcal{Z}$ are plotted. On the left, the networks are trained with $|\mathcal{D}| = 500$, and on the right, $|\mathcal{D}| = 10000$. In both settings, the networks have 1500 neurons in the hidden layer, are trained in domain $[0, 8]^2$, and are tested with $|\mathcal{Z}| = 10000$.

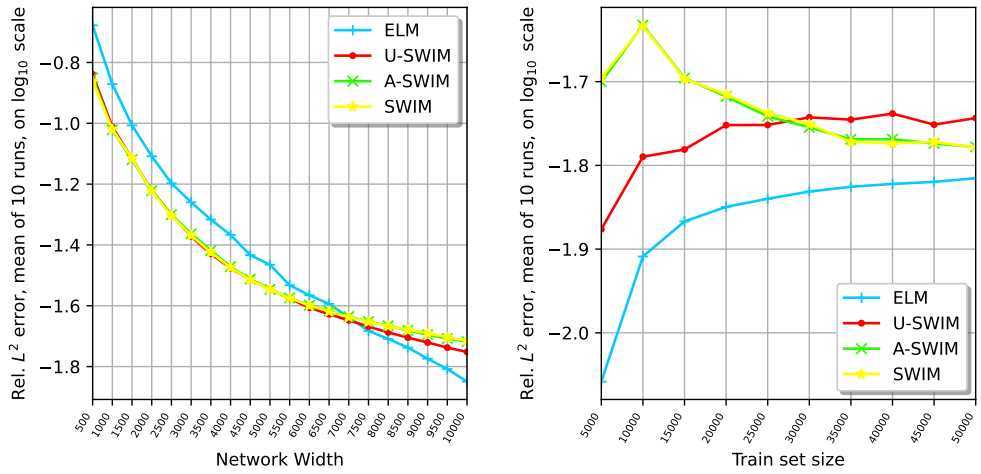


Figure 13: For double pendulum in Eq. (3.19) gradient errors on $\mathcal{D}$ are plotted. The left plot is network width scaling with $|\mathcal{D}| = 20000$. The right plot is $|\mathcal{D}|$ scaling with network width set to 10000. Networks in both settings are trained in domain $[-\pi, \pi] \times [-1, 1]$.

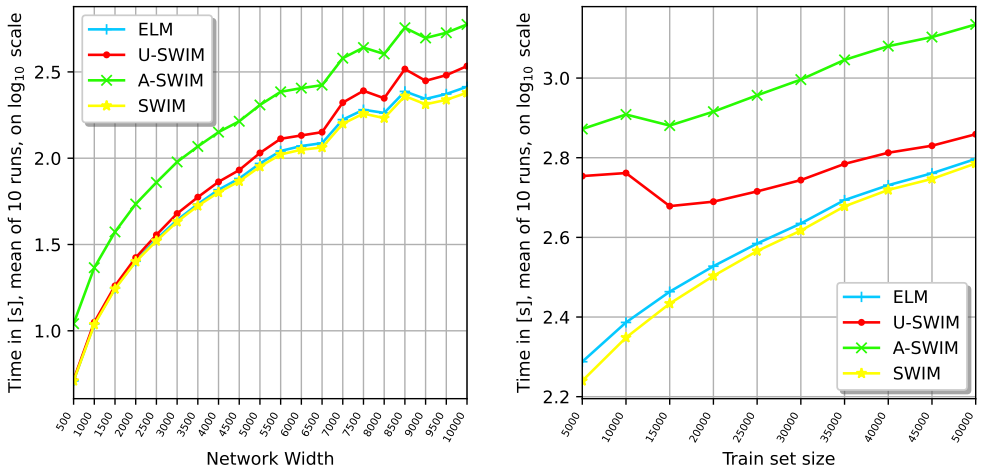


Figure 14: For double pendulum in Eq. (3.19) train times in seconds are plotted. The left plot is network width scaling with $|\mathcal{D}| = 20000$ points. The right plot is $|\mathcal{D}|$ scaling with network width set to 10000. Similar to Appendix Fig. 1, the large train times at low train set sizes may indicate duplicate points sampled for the weight sampling algorithm, which needed to be re-sampled.

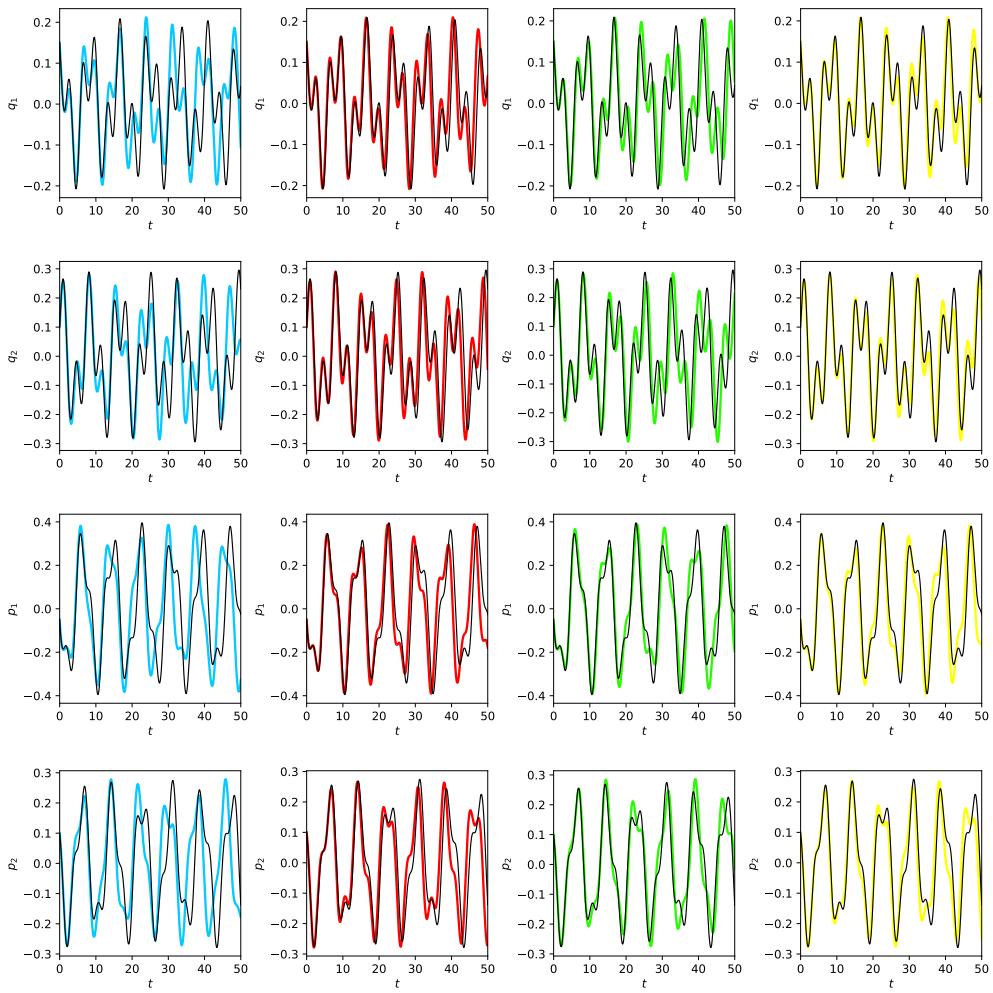


Figure 15: For double pendulum in Eq. (3.19) integration of the position and momenta are plotted over time with the same settings as in Fig. 3.11. From the leftmost column to the right: ELM, U-SWIM, A-SWIM, and SWIM predictions are plotted over the ground truth (black line). The ground truth represents symplectic Euler using the true Hamiltonian.

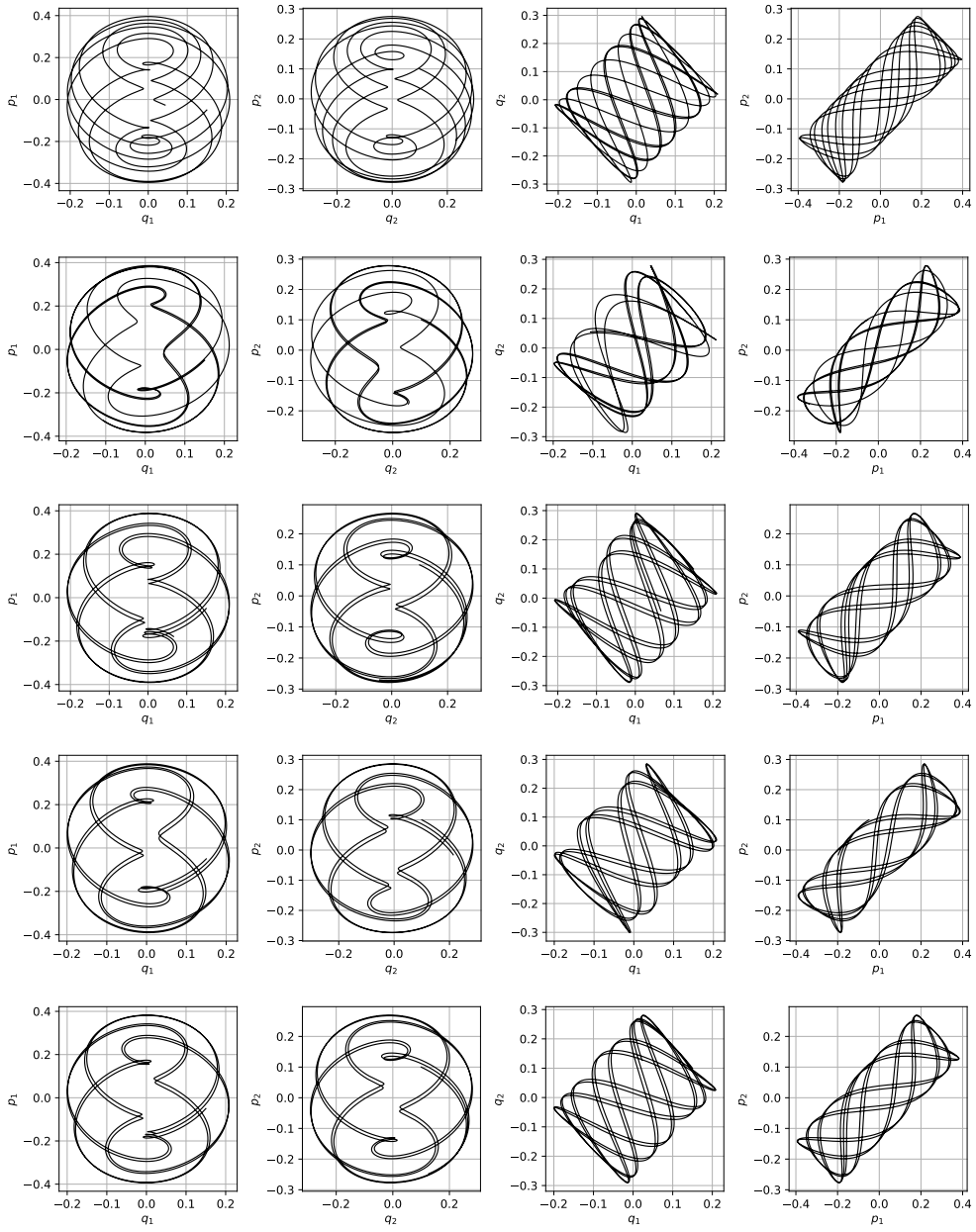


Figure 16: For double pendulum in Eq. (3.19) Poincaré plots are displayed with the same settings as in Fig. 3.11. The top row is the ground truth, representing symplectic Euler using the true Hamiltonian. Beginning from the second row to the bottom row: ELM, U-SWIM, A-SWIM, and SWIM predictions are plotted.

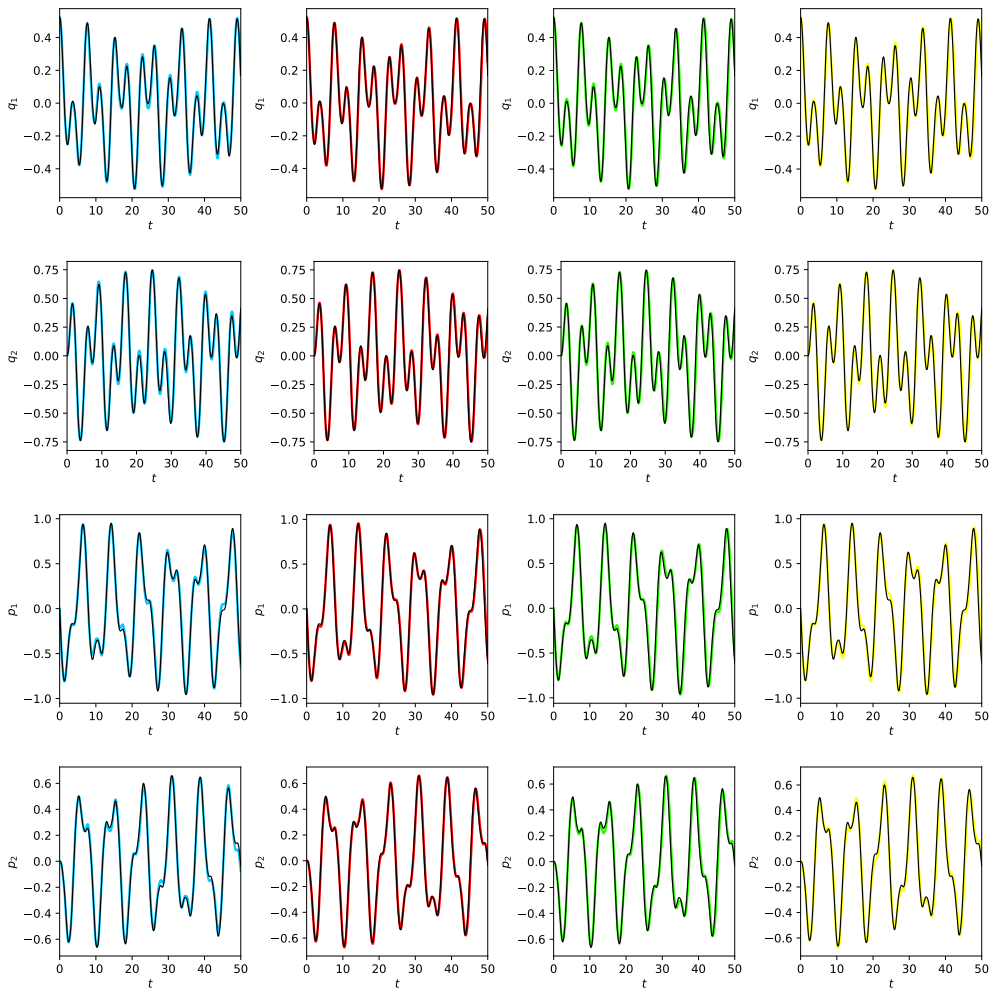


Figure 17: For double pendulum in Eq. (3.19) integration of the position and momenta are plotted over time with the same settings as in Fig. 3.12. From the leftmost column to the right: ELM, U-SWIM, A-SWIM, and SWIM predictions are plotted over the ground truth (black line). The ground truth represents symplectic Euler using the true Hamiltonian.

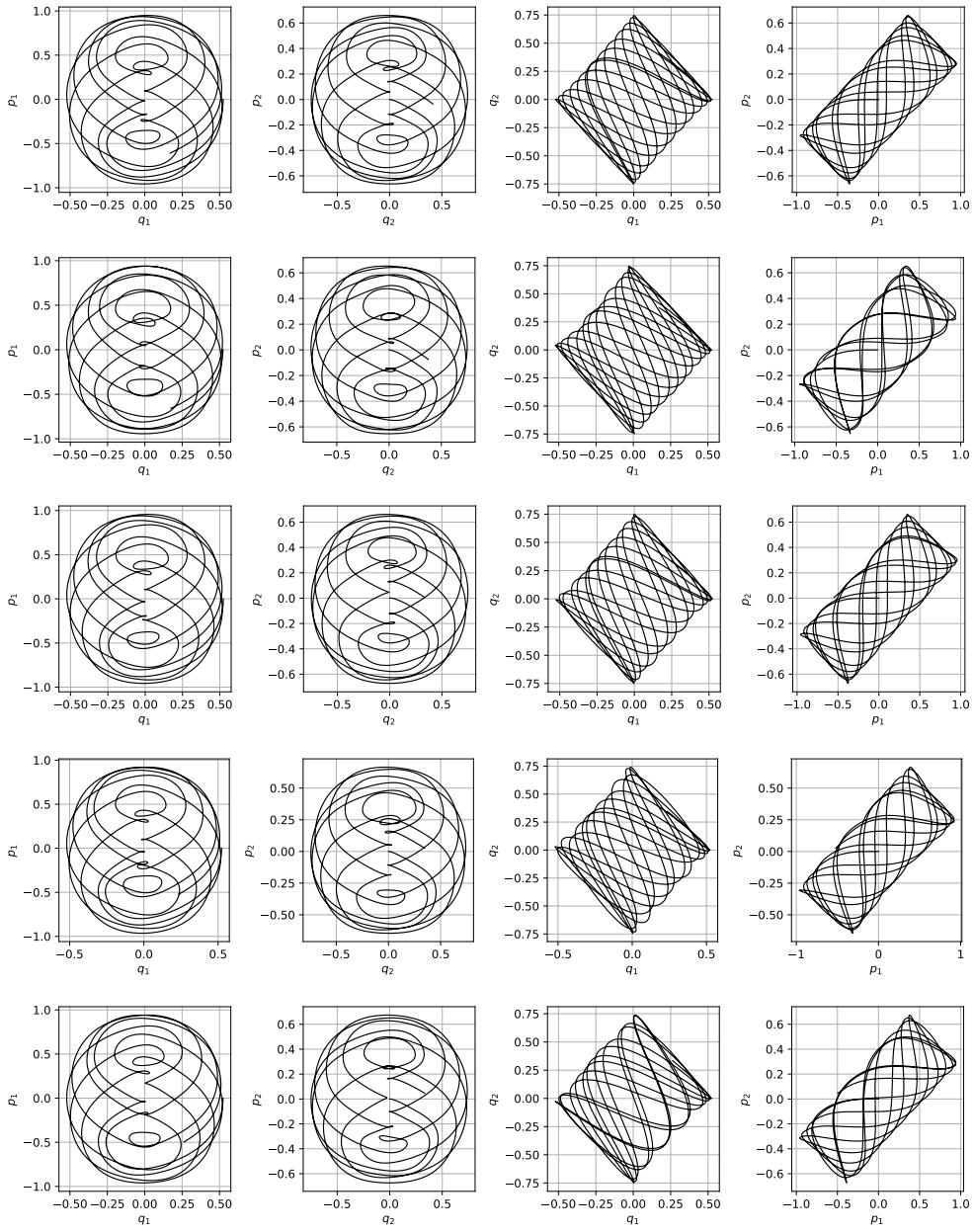


Figure 18: For double pendulum in Eq. (3.19) Poincaré plots are displayed with the same settings as in Fig. 3.12. The top row is the ground truth, representing symplectic Euler using the true Hamiltonian. Beginning from the second row to the bottom row: ELM, U-SWIM, A-SWIM, and SWIM predictions are plotted.

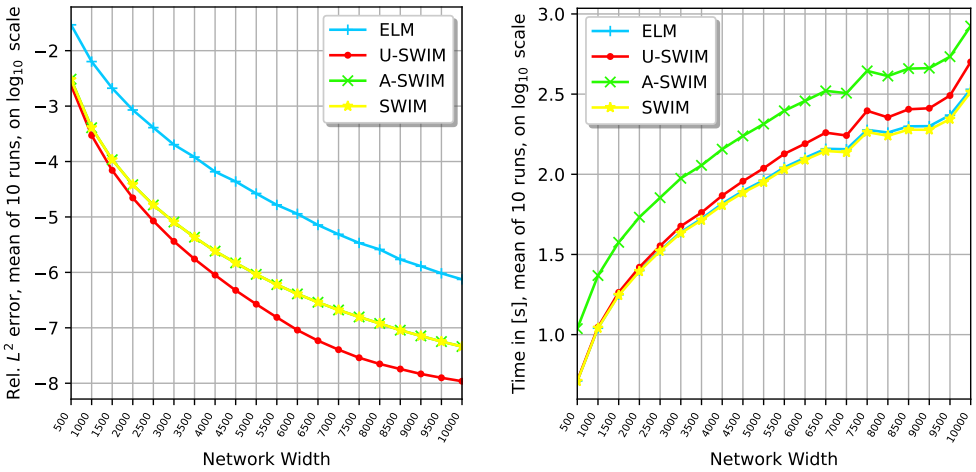


Figure 19: For Hénon-Heiles in Eq. (3.20) with $\alpha = 1$, gradient errors on $\mathcal{D}$ are plotted. Network width scaling with $|\mathcal{D}| = 20000$ is displayed in the left plot, and the corresponding times taken to train the networks are displayed on the right. Networks are trained in domain $[-1, 1]^2$.

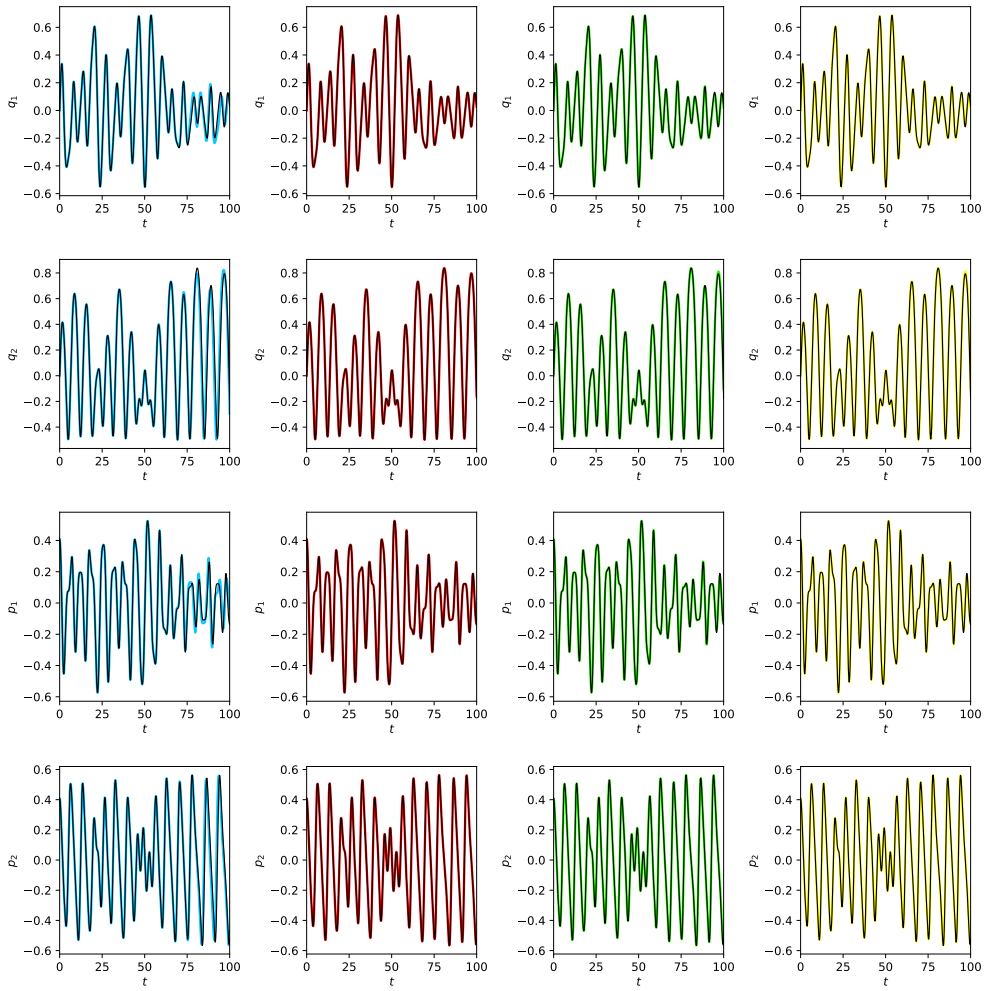


Figure 20: For Hénon-Heiles in Eq. (3.20) with $\alpha = 1$, integration of the position and momenta are plotted over time with the same settings as in Fig. 3.14. From the leftmost column to the right: ELM, U-SWIM, A-SWIM, and SWIM predictions are plotted over the ground truth (black line). The ground truth represents symplectic Euler using the true Hamiltonian.

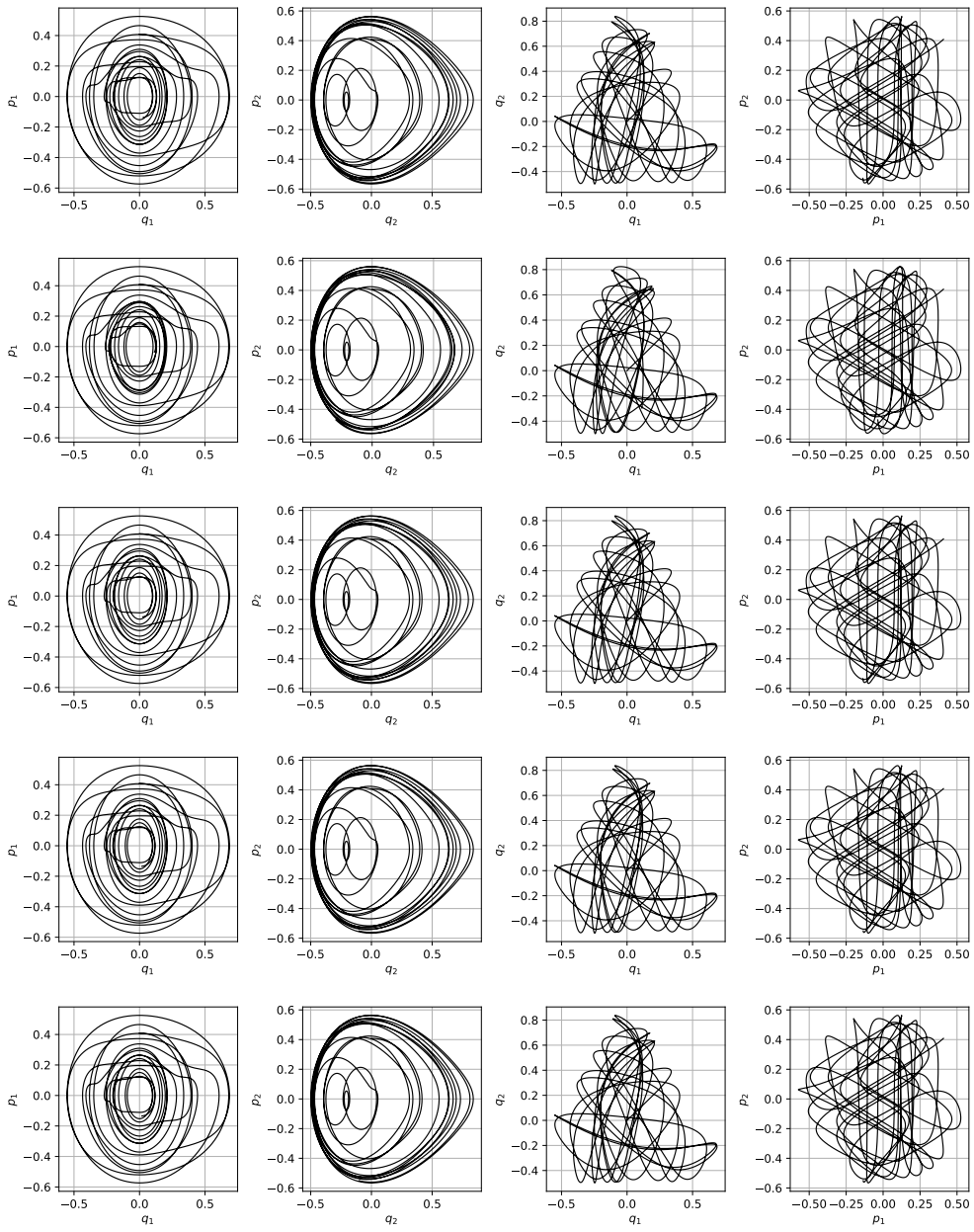


Figure 21: For Hénon-Heiles in Eq. (3.20) with $\alpha = 1$ Poincaré plots are displayed with the same settings as in Fig. 3.14. The top row is the ground truth, representing symplectic Euler using the true Hamiltonian. Beginning from the second row to the bottom row: ELM, U-SWIM, A-SWIM, and SWIM predictions are plotted.

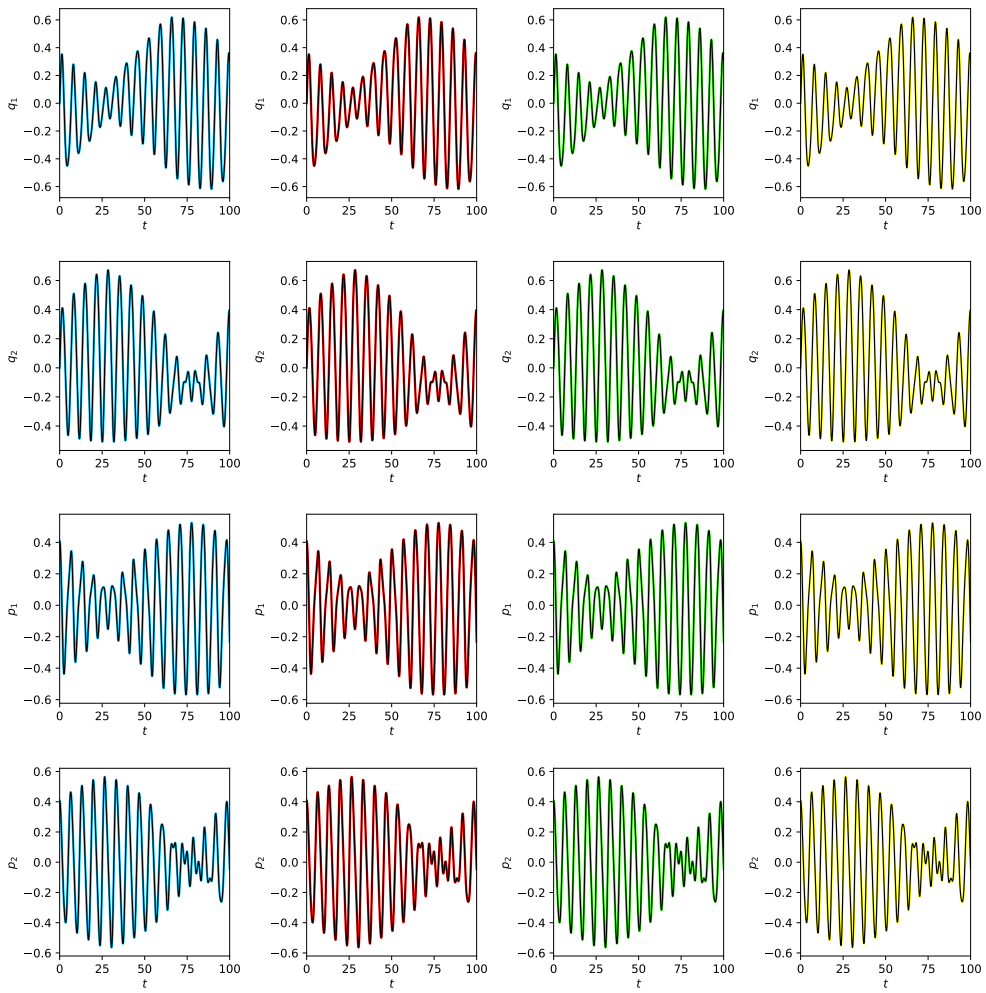


Figure 22: For Hénon-Heiles in Eq. (3.20) with $\alpha = 0.7$ integration of the position and momenta are plotted over time with the same settings as in Fig. 3.14 except the α value. From the leftmost column to the right: ELM, U-SWIM, A-SWIM, and SWIM predictions are plotted over the ground truth (black line). The ground truth represents symplectic Euler using the true Hamiltonian.

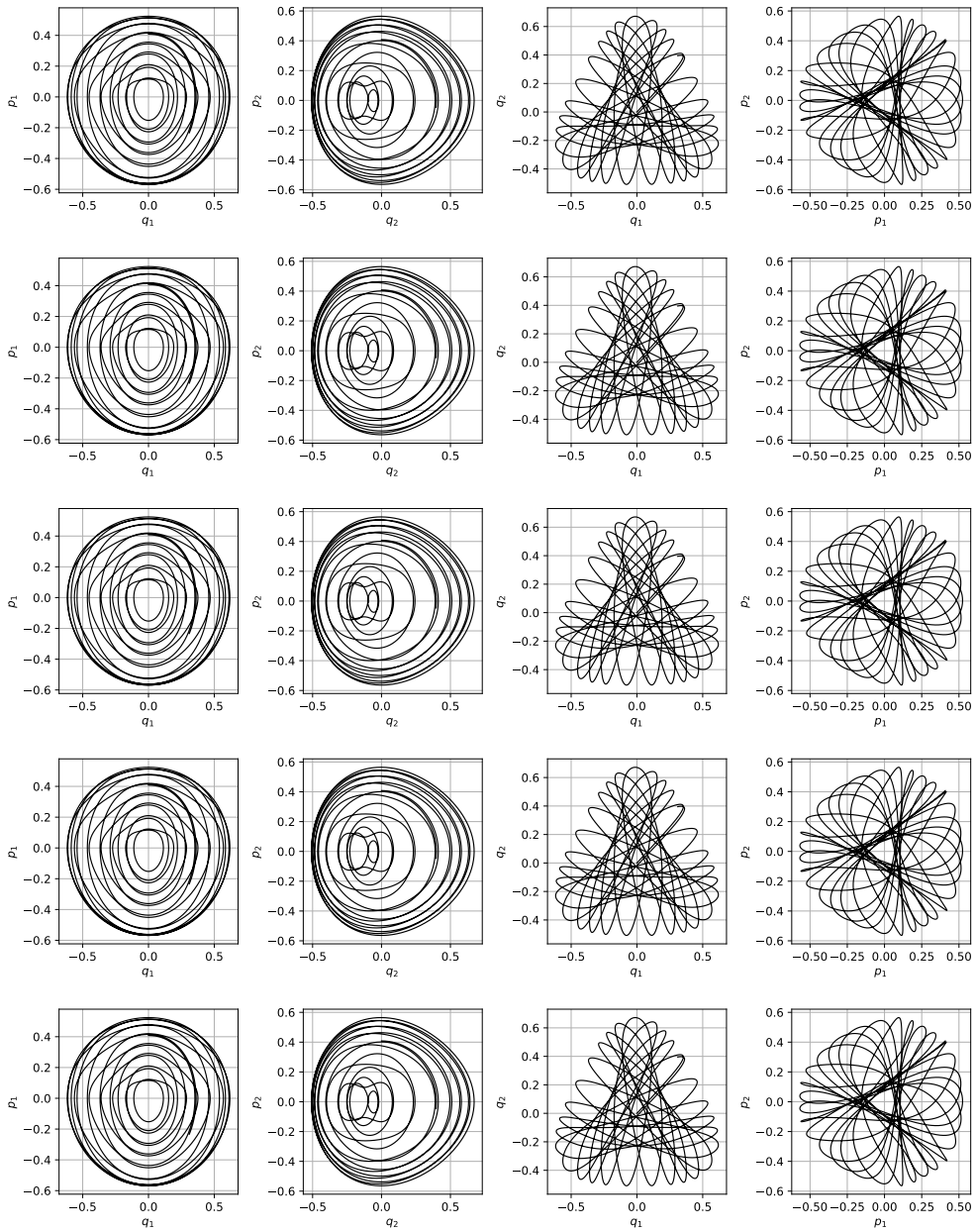


Figure 23: For Hénon-Heiles in Eq. (3.20) with $\alpha = 0.7$ Poincaré plots are displayed with the same settings as in Fig. 3.14 except the α value. The top row is the ground truth, representing symplectic Euler using the true Hamiltonian. Beginning from the second row to the bottom row: ELM, U-SWIM, A-SWIM, and SWIM predictions are plotted.