Training Hamiltonian neural networks without backpropagation

Atamert Rahma[1] **Chinmay Datar**[1,2] **Felix Dietrich**[1,3]
[1]School of Computation, Information and Technology
[2] Institute for Advanced Study
[3] Munich Data Science Institute
Technical University of Munich
Munich, Germany
{atamert.rahma,chinmay.datar,felix.dietrich}@tum.de

Abstract

Neural networks that synergistically integrate data and physical laws offer great promise in modeling dynamical systems. However, iterative gradient-based optimization of network parameters is often computationally expensive and suffers from slow convergence. In this work, we present a backpropagation-free algorithm to accelerate the training of neural networks for approximating Hamiltonian systems through data-agnostic and data-driven algorithms. We empirically show that data-driven sampling of the network parameters outperforms data-agnostic sampling or the traditional gradient-based iterative optimization of the network parameters when approximating functions with steep gradients or wide input domains. We demonstrate that our approach is more than 100 times faster with CPUs than the traditionally trained Hamiltonian Neural Networks using gradient-based iterative optimization and is more than four orders of magnitude accurate in chaotic examples, including the Hénon-Heiles system.

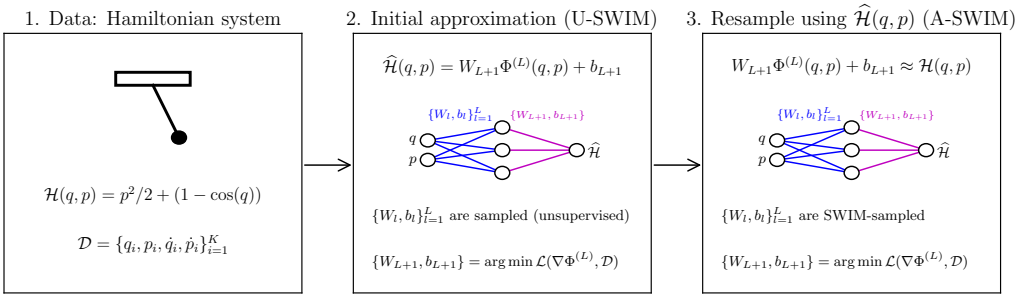

Figure 1: Approximate-SWIM (A-SWIM) algorithm: This figure illustrates the process of approximating a Hamiltonian system from data, including generalized "position" q and "momentum" p coordinates, along with their time derivatives $\dot{q}$ and $\dot{p}$. **Left**: The given Hamiltonian system. **Center**: Sampling hidden layer weights and biases $\{W_l, b_l\}_{l=1}^L$ in the unsupervised setting. **Right**: Resampling hidden layer parameters using the approximated function values $\widehat{\mathcal{H}}(q,p)$ obtained in stage two. Note that in steps two and three, the linear layer parameters $\{W_{L+1}, b_{L+1}\}$ are optimized by solving a linear least-squares problem.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

1 Introduction

Learning techniques often integrate a series of inductive biases to learn inherent patterns within data and extend the generalization capability beyond the training set. In the context of approximating physical systems, physical priors are crucial for capturing the system's nature, including its dynamics and underlying physical laws [32, 7, 3]. In particular, **Hamiltonian Neural Networks (HNNs)** leverage Hamilton's equations [14, 15] and re-formulation of the loss functions to learn the conserved quantities of target systems [12, 1]. Subsequent research has introduced various extensions to HNNs, such as port-HNNs [8] for controlling forces and dissipative HNNs [30] for modeling dissipative systems. Recent work has further extended HNNs by integrating symplectic methods [9, 25, 24]. Symplectic Recurrent Neural Networks (SRNNs), proposed by Chen et al. [4], outperform traditional HNNs in long-term trajectory prediction from time series data by incorporating symplectic integrators. Similarly, Symplectic Ordinary Differential Equations (SymODEN) [36] and its dissipative variant [35] use symplectic integrators and can also account for external forces, such as those involved in control systems. Additionally, Xiong et al. [33] employed explicit high-order integrator schemes within an extended phase space to address non-separable Hamiltonians.

Despite the success, many challenges are becoming apparent, mainly stemming from gradient-based optimization of parameters using backpropagation. Backpropagation through an integrator is computationally expensive and time-consuming [33]. Moreover, iterative gradient-descent-based training approaches pose a significant challenge for traditional neural networks due to their slow convergence rates [21]. Jakovac et al. [22] explored **random feature models** in Hamiltonian flow approximation and reported faster training times, although their work was limited to the data-agnostic method: Extreme Learning Machine (ELM) [18]. We adopt a strategy that is similar to that of Bertalan et al. [1], solving system identification through a linear PDE, but replacing their Gaussian Process ansatz with data-driven random features. Our key contributions in this work are as follows:

1. We explore data-agnostic [18] and data-driven algorithms [2, 5] to compute parameters of the neural networks for learning Hamiltonian functions from data **without backpropagation**.
2. We investigate the **benefits of data-driven sampling** over data-agnostic approaches in improving accuracy for systems like single/double pendulums, Lotka-Volterra, and Hénon-Heiles.
3. We demonstrate that data-driven sampling achieves high accuracy in the unsupervised setting **with limited data** using approximate initial function values.

2 Method

We consider an autonomous Hamiltonian on Euclidean space $E = \mathbb{R}^{2d}$, where $d \in \mathbb{N}$ is the number of degrees of freedom of the underlying system. The Hamiltonian is defined as $\mathcal{H} : E \to \mathbb{R}$ and incorporates the laws of motion defined through Hamilton's equations [14, 15]

$$J \cdot \nabla \mathcal{H}(q,p) - v(q,p) = \vec{0} \tag{1}$$

for all $\begin{bmatrix} q \\ p \end{bmatrix} \in E$, where $J = \begin{bmatrix} 0_d & I_d \\ -I_d & 0_d \end{bmatrix} \in \{0,1\}^{2d \times 2d}$, $I_d \in \{0,1\}^{d \times d}$ is the identity matrix, 0_d is the d by d square matrix of zeros, and $v = [\dot{q} \quad \dot{p}]^\mathsf{T}$ is the vector field on the phase space E which only depends on generalized "position" $q(t) \in \mathbb{R}^d$ and "momentum" $p(t) \in \mathbb{R}^d$ coordinates.

Sampling neural networks to approximate Hamiltonian functions: We model the Hamiltonian that we would like to approximate using a fully-connected neural network Φ following Definition A.1 with L hidden layers given data $\mathcal{D} = \{q_i, p_i, \dot{q}_i, \dot{p}_i\}_{i=1}^K$, such that

$$\widehat{\mathcal{H}}(q,p) = \Phi(q,p) = W_{L+1}\Phi^{(L)}(q,p) + b_{L+1} \overset{!}{=} \mathcal{H}(q,p) \text{ for all } [q \quad p]^\mathsf{T} \in E, \tag{2}$$

where $\{W_l, b_l\}_{l=1}^L$ are the weights and biases of the hidden layers, W_{L+1} and b_{L+1} are the weights and biases of the last linear layer, and we write $\Phi^{(l)}(\cdot)$ to represent the output of the l-th layer of the network. To construct the hidden layers we sample the parameters $\{W_l, b_l\}_{l=1}^L$ using the following data-agnostic and data-driven sampling schemes.

Data-agnostic sampling: We use the **"Extreme Learning Machine" (ELM)** approach which is well-studied [29, 26, 18, 18, 34, 23, 22] along with the error bounds and approximation capabilities [19, 28, 34, 23]. In this approach, we sample the weights $\{W_l\}_{l=1}^{L}$ from the standard normal distribution and biases $\{b_l\}_{l=1}^{L}$ from the uniform distribution for all hidden layers.

Data-driven sampling: We use the **"Sample Where It Matters" (SWIM)** algorithm by Bolager et al. [2]. Here, each weight and bias pair in the hidden layers is constructed using a data point pair sampled from the input space so that $w_{l,i} = s_1 \frac{x^{(2)}_{l-1,i} - x^{(1)}_{l-1,i}}{\|x^{(2)}_{l-1,i} - x^{(1)}_{l-1,i}\|^2}$ and $b_{l,i} = -\langle w_{l,i}, x^{(1)}_{l-1,i} \rangle - s_2$, where $(w_{l,i}, b_{l,i})$ are the i-th row of the parameters in the l-th hidden layer, $(x^{(1)}_{l-1,i}, x^{(2)}_{l-1,i})$ are the outputs of the hidden layer $(l-1)$ of the data point pair sampled for constructing the parameters $(w_{l,i}, b_{l,i})$, and (s_1, s_2) are constants depending on the activation function used in the associated hidden layer. SWIM-sampled neural networks are discussed in more detail in Definition A.2.

The construction associates each hidden layer parameter with an input pair sampled from the given data points from the input space $\mathcal{X}$. In the unsupervised setting, we sample the pairs of points uniformly randomly and refer to this as **Uniform-SWIM (U-SWIM)**. In the supervised setting, we sample the points with a density proportional to the finite differences $\|\mathcal{H}(x^{(2)}_0) - \mathcal{H}(x^{(1)}_0)\| / \|x^{(2)}_{l-1} - x^{(1)}_{l-1}\|$ and refer to this as **SWIM** [2]. The intuition behind the density in SWIM is to place more basis functions in the part of the space domain where the gradient of the underlying function is large. To retain this efficient placement of basis functions using SWIM, but in the unsupervised setting, we propose an adaptive approach in which we first compute an initial approximation and then use the SWIM algorithm to re-sample the network parameters as illustrated in Figure 1. We refer to this method as **Approximate-SWIM (A-SWIM)** and use U-SWIM for the initial approximation.

After sampling all the hidden layer parameters, we discuss finding the optimal parameters for the last linear layer of the network. Differentiating Equation (2) with respect to input x results in $W_{L+1} \nabla \Phi^{(L)}(x) \overset{!}{=} \nabla \mathcal{H}(x) \overset{\text{eq. (1)}}{=} J^{-1} \dot{x}$ for a single data point $x = [q \quad p]^\mathsf{T} \in E$. Similar to [1], we set up a fully linear system using the given data, where we replace the basis functions of the approximation with the outputs of the last hidden layer of the network Φ:

$$\underbrace{\begin{bmatrix} \nabla \Phi^{(L)}(x_1) & \cdots & \nabla \Phi^{(L)}(x_K) & \Phi^{(L)}(x_0) \\ 0 & \cdots & 0 & 1 \end{bmatrix}^\mathsf{T}}_{A \in \mathbb{R}^{(2dK+1) \times (N_L+1)}} \cdot \underbrace{\begin{bmatrix} W_{L+1}^\mathsf{T} \\ b_{L+1} \end{bmatrix}}_{w \in \mathbb{R}^{N_L+1}} \overset{!}{=} \underbrace{\begin{bmatrix} J^{-1}\dot{x}_1 & \cdots & J^{-1}\dot{x}_K & \mathcal{H}(x_0) \end{bmatrix}^\mathsf{T}}_{u \in \mathbb{R}^{2dK+1}}, \tag{3}$$

where we write N_l for the width of the l-th hidden layer. We note that we assume we know the true Hamiltonian value $\mathcal{H}(x_0)$ for a single data point $x_0 \in E$ to fix the integration constant b_{L+1} (which is only a scalar). We also assume we know the true time derivatives $\dot{x}$ in this work to avoid discretization errors. We shortly discuss error correction techniques in Appendix B if finite differences are used to retrieve $\dot{x}$, e.g. from trajectory data. Equation (3) gives rise to a well-studied convex optimization problem which can be solved by using the linear least squares $\begin{bmatrix} W_{L+1}^\mathsf{T} & b_{L+1} \end{bmatrix}^\mathsf{T} = \arg\min_w \|Aw - u\|^2$.

The details of the algorithms for constructing sampled HNNs using the samplers ELM, U-SWIM, or A-SWIM are provided in Algorithm A.1. The gradient $\nabla \Phi^{(L)}$ is computed analytically as we use differentiable activation functions ($\sigma = \tanh$).

3 Numerical results and discussion

In this section, we report our numerical results, including approximations of different Hamiltonian systems with different configurations. Details regarding the experiment setup, used Hamiltonian target functions, software versions, and the evaluated error function are explained in Appendix B. The code is available open-source at `https://github.com/AlphaGergedan/Sampling-HNNs`.

Single pendulum and Lotka-Volterra experiments: Table 1 summarizes the experiments for the single pendulum and Lotka-Volterra [10] Hamiltonian functions. The network width is scaled for

each experiment in the table, and Figure B.3 and Figure B.4 describe how the error decays with the network width. Firstly, we observe that the losses obtained with gradient-free training are 3-9 orders of magnitude better. Secondly, we observe that as the domain size or the frequency of the single pendulum increases, SWIM is usually more accurate and also needs fewer neurons to attain a certain accuracy compared to ELM as it exploits the data to place basis functions efficiently (more where the solution gradient is large). Thirdly, the accuracies obtained with A-SWIM are much better than those with U-SWIM and are very close to those obtained with SWIM, even though the true values of the Hamiltonian are not available in A-SWIM. Lastly, Figure 2 demonstrates that increasing the frequency parameter in the single-pendulum experiment leads to steep solution gradients, where A-SWIM and SWIM prove to be far more accurate than ELM.

Table 1: Single pendulum and Lotka-Volterra approximations are summarized. In all the listed experiments, A-SWIM and SWIM errors have the same values up to the order that we list here therefore we list them together in the column (A-)SWIM. Network width is set to 1000, domain and other model parameters are set according to Table B.5 and Table B.3, respectively.

System name	Hamiltonian	Domain	HNN	ELM	(A-)SWIM
Single pendulum	eq. (B.4)	$[-2\pi, 2\pi] \times [-1, 1]$	2.17E-03	**1.49E-11**	3.62E-10
Single pendulum	eq. (B.4)	$[-2\pi, 2\pi] \times [-6, 6]$	7.37E-04	9.82E-08	**1.55E-09**
Lotka-Volterra	eq. (B.6)	$[-2, 2] \times [-2, 2]$	2.35E-03	**3.04E-12**	1.48E-10
Lotka-Volterra	eq. (B.6)	$[-5, 5] \times [-5, 5]$	1.38E-03	1.02E-08	**7.99E-09**
Lotka-Volterra	eq. (B.7)	$[0, 8] \times [0, 8]$	2.63E-03	3.27E-06	**1.51E-08**

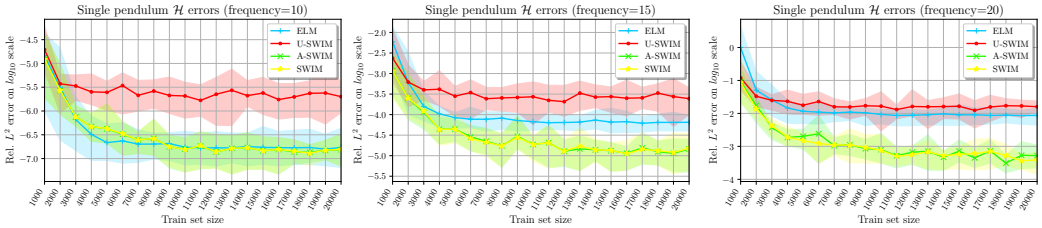

Figure 2: Single pendulum (with frequency parameter) approximation errors are plotted.

Chaotic systems: We summarize our experiments with the double pendulum and Hénon-Heiles [17] systems in Table 2. The key observation is that the gradient-free training with A-SWIM is more than 100 times faster (with CPUs) compared to the gradient-based optimization of HNNs for the same error in the double pendulum experiment and four orders of magnitude lower error in the Hénon-Heiles experiment. Moreover, if the initial approximation with (U-SWIM) is sufficiently accurate, A-SWIM can closely match the SWIM method's performance. We emphasize that while SWIM uses true Hamiltonian values for sampling, A-SWIM relies solely on approximate values for this process, which results in approximately double the training time.

Table 2: Summary of results for chaotic system approximations. Note that we report the CPU time. Please refer to Table B.4 and Table B.5 for details on models.

	Double pendulum		Hénon-Heiles	
Method	Training time (s)	Rel. L^2 error	Training time (s)	Rel. L^2 error
HNN	10485.4	**3.62E-03**	13140.6	6.68E-04
ELM	43.1	5.69E-03	46.0	2.07E-02
U-SWIM	**39.6**	5.00E-03	**41.7**	8.22E-08
A-SWIM	85.2	4.08E-03	89.4	**6.80E-08**
SWIM	40.0	4.18E-03	41.9	**6.80E-08**

Conclusion and future work: We presented a framework for approximating Hamiltonian functions using sampled neural networks without requiring iterative parameter optimization via backpropagation. Our approach (A-SWIM) is more than two orders of magnitude faster to train on CPUs than gradient-based optimization of HNNs in the chaotic systems we consider, and it achieves more than four orders of magnitude greater accuracy in most examples. Our evaluation demonstrates that data-driven sampling via SWIM outperforms data-agnostic methods when the function being approximated has steep gradients or large input domains. We note that our approach requires solving a large linear system. In higher dimensional examples that require a lot of computational requirements, one can rely on HPC resources and iterative solvers. In the future, we intend to extend this work to dissipative systems, as this work assumes symplecticity, while most real-world systems are dissipative. Another important direction is to extend our algorithm to handle noisy data. Lastly, the longer training times reported in [33] may be compensated by utilizing sampled networks to learn the flow map of Hamiltonian systems where using backpropagation is computationally expensive.

Acknowledgments and Disclosure of Funding

The authors gratefully acknowledge the computational and data resources provided by the Leibniz Supercomputing Centre (www.lrz.de). We also thank the anonymous reviewers at NeurIPS for their constructive feedback. F.D. acknowledges funding by the German Research Foundation—project 468830823, and association to DFG-SPP-229. C.D. is partially funded by the Institute for Advanced Study (IAS) at the Technical University of Munich.

References

[1] Tom Bertalan, Felix Dietrich, Igor Mezic, and Ioannis G. Kevrekidis. On learning hamiltonian systems from data. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 29(12), 2019. doi: 10.1063/1.5128231.

[2] Erik L Bolager, Iryna Burak, Chinmay Datar, Qing Sun, and Felix Dietrich. Sampling weights of deep neural networks. In *Advances in Neural Information Processing Systems*, volume 36, pages 63075–63116. Curran Associates, Inc., 2023.

[3] Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. A compositional object-based approach to learning physical dynamics. *arXiv*, 2016.

[4] Zhengdao Chen, Jianyu Zhang, Martin Arjovsky, and Leon Bottou. Symplectic recurrent neural networks. *arXiv*, 2019.

[5] Chinmay Datar, Taniya Kapoor, Abhishek Chandra, Qing Sun, Iryna Burak, Erik Lien Bolager, Anna Veselovska, Massimo Fornasier, and Felix Dietrich. Solving partial differential equations with sampled neural networks. *arXiv*, May 2024.

[6] Marco David and Florian Mehats. Symplectic learning for hamiltonian neural networks. *Journal of Computational Physics*, 494:112495, 2023.

[7] Filipe de Avila Belbute-Peres, Kevin Smith, Kelsey Allen, Josh Tenenbaum, and J Zico Kolter. End-to-end differentiable physics for learning and control. *Advances in neural information processing systems*, 31, 2018.

[8] Shaan A Desai, Marios Mattheakis, David Sondak, Pavlos Protopapas, and Stephen J Roberts. Port-hamiltonian neural networks for learning explicit time-dependent dynamical systems. *Physical Review E*, 104(3):034312, 2021.

[9] Eva Dierkes, Christian Offen, Sina Ober-Bloebaum, and Kathrin Flasskamp. Hamiltonian neural networks with automatic symmetry detection. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 33(6): 063115, 2023. doi: 10.1063/5.0142969.

[10] Rui Loja Fernandes and Waldyr Muniz Oliva. Hamiltonian dynamics of the lotka-volterra equations. In *International Conference on Differential Equations, Lisboa*, pages 327–334. World Scientific, 1995.

[11] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[12] Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. *Advances in neural information processing systems*, 32, 2019.

[13] Ernst Hairer, Marlis Hochbruck, Arieh Iserles, and Christian Lubich. Geometric numerical integration. *Oberwolfach Reports*, 3(1):805–882, 2006.

[14] William Rowan Hamilton. On a general method in dynamics. *Philosophical Transactions of the Royal Society*, 124:247–308, 1834.

[15] William Rowan Hamilton. Second essay on a general method in dynamics. *Philosophical Transactions of the Royal Society*, 125:95–144, 1835.

[16] Charles R. Harris, K. Jarrod Millman, Stefan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernandez del Rio, Mark Wiebe, Pearu Peterson, Pierre Gerard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825): 357–362, 2020. doi: 10.1038/s41586-020-2649-2.

[17] Michel Henon and Carl Heiles. The applicability of the third integral of motion: some numerical experiments. *Astronomical Journal, Vol. 69, p. 73 (1964)*, 69:73, 1964.

[18] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: a new learning scheme of feedforward neural networks. In *2004 IEEE international joint conference on neural networks (IEEE Cat. No. 04CH37541)*, volume 2, pages 985–990, 2004.

[19] Guang-Bin Huang, Lei Chen, and Chee Siew. Universal approximation using incremental constructive feedforward networks with random hidden nodes. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 17:879–92, 2006. doi: 10.1109/TNN.2006.875977.

[20] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi: 10.1109/MCSE.2007.55.

[21] Robert A Jacobs. Increased rates of convergence through learning rate adaptation. *Neural networks*, 1(4): 295–307, 1988.

[22] Antal Jakovac, Marcell T Kurbucz, and Péter Posfay. Reconstruction of observed mechanical motions with artificial intelligence tools. *New Journal of Physics*, 24(7):073021, 2022. doi: 10.1088/1367-2630/ac7c2d.

[23] Ho Chun Leung, Chi Sing Leung, and Eric Wing Ming Wong. Fault and noise tolerance in the incremental extreme learning machine. *IEEE Access*, 7:155171–155183, 2019.

[24] Sina Ober-Bloebaum and Christian Offen. Variational learning of Euler–Lagrange dynamics from data. *Journal of Computational and Applied Mathematics*, 421:114780, 2023. doi: 10.1016/j.cam.2022.114780.

[25] C. Offen and S. Ober-Bloebaum. Symplectic integration of learned Hamiltonian systems. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 32(1):013122, 2022. doi: 10.1063/5.0065913.

[26] Y-H Pao and Yoshiyasu Takefuji. Functional-link net computing: theory, system architecture, and functionalities. *Computer*, 25(5):76–79, 1992.

[27] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[28] Ali Rahimi and Benjamin Recht. Uniform approximation of functions with random bases. In *2008 46th annual allerton conference on communication, control, and computing*, pages 555–561. IEEE, 2008.

[29] Wouter F Schmidt, Martin A Kraaijveld, Robert PW Duin, et al. Feed forward neural networks with random weights. In *International conference on pattern recognition*, pages 1–1. IEEE Computer Society Press, 1992.

[30] Andrew Sosanya and Sam Greydanus. Dissipative hamiltonian neural networks: Learning dissipative and conservative dynamics separately. *arXiv*, 2022.

[31] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stefan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, Ilhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald,

Antonio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

[32] Nicholas Watters, Daniel Zoran, Theophane Weber, Peter Battaglia, Razvan Pascanu, and Andrea Tacchetti. Visual interaction networks: Learning a physics simulator from video. *Advances in neural information processing systems*, 30, 2017.

[33] Shiying Xiong, Yunjin Tong, Xingzhe He, Shuqi Yang, Cheng Yang, and Bo Zhu. Nonseparable symplectic neural networks. *arXiv*, 2022.

[34] Rui Zhang, Yuan Lan, Guang-Bin Huang, and Zong-Ben Xu. Universal approximation of extreme learning machine with adaptive growth of hidden nodes. *IEEE Transactions on Neural Networks and Learning Systems*, 23(2):365–371, 2012. doi: 10.1109/TNNLS.2011.2178124.

[35] Yaofeng Desmond Zhong, Biswadip Dey, and Amit 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.

[36] Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Symplectic ode-net: Learning hamiltonian dynamics with control. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia*, 2020.

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

Appendix

A Mathematical framework

Feed-forward neural networks: In this paper, we work with feed-forward neural networks configured for regression, i.e., no activation is used in the output layer to approximate a Hamiltonian. We define the notation used for neural networks in this work in Definition A.1.

Definition A.1. Let $\mathcal{X} \subseteq \mathbb{R}^D$ be an input space, and $\mathcal{Y} \subseteq \mathbb{R}$ a one-dimensional output space. We write

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

as the output of the l-th layer of a network Φ with L hidden layers, where

- $\sigma : \mathbb{R} \to \mathbb{R}$ is an activation function applied element-wise,
- $\{W_l, b_l\}_{l=1}^{L+1}$ are the parameters of 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 l-th layer with $N_0 = D$ and $N_{L+1} = 1$.

We write $\Phi(x) \in \mathcal{Y}$ for the network output given a single data point $x \in \mathcal{X}$ and $\Phi(X) \in \mathcal{Y}^K$ given an input matrix $X \in \mathcal{X}^{K \times D}$.

"Sample Where It Matters" (SWIM): Our work is closely related to [2] as we use their SWIM algorithm for the data-driven sampling of the hidden layer parameters. In Definition A.2 we summarize a SWIM-sampled network from [2] to provide the overall idea of using data to sample the hidden layer parameters.

Definition A.2. Let Φ be a network as defined in Definition A.1. For $l = 1, \ldots, L$ let $x_{0,i}^{(1)}, x_{0,i}^{(2)}$ be pairs of points sampled over $\mathcal{X} \times \mathcal{X}$. If the weights and biases of each layer $l = 1, 2, \ldots, L$ and neuron $i = 1, 2, \ldots, N_l$ have the form

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

then we say the hidden layer parameters $\{W_l, b_l\}_{l=1}^{L}$ are SWIM-sampled and Φ is a SWIM-sampled network. $\|\cdot\|$ 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 every input pair $\{x^{(1)}, x^{(2)}\}$. For $\tanh$, the only activation function that we use in our numerical experiments, $s_1 = 2s_2$ and $s_2 = {}^{\ln(3)}/_2$ are set, which implies $\sigma(x^{(1)}) = {}^1/_2$ and $\sigma(x^{(2)}) = {}^{-1}/_2$, and $\sigma\big({}^{(x^{(1)}+x^{(2)})}/_2\big) = 0$,
- $x_{l-1,i}^{(k)} = \Phi^{(l-1)}(x_{0,i}^{(k)})$ for $k \in \{1, 2\}$ and $x_{l-1,i}^{(1)} \neq x_{l-1,i}^{(2)}$,
- we write $w_{l,i}$ for the i-th row of W_l and $b_{l,i}$ for the i-th entry of b_l.

The authors also provide a probability distribution for the data points in the input space $\mathcal{X}$ (see Definition A.3) to sample where it matters: at large gradients. The supervised SWIM algorithm we evaluate utilizes this distribution.

Definition A.3. Let $\mathcal{X}$ be an input space and let $\mathcal{Y} = f(\mathcal{X})$ be the function space given a function $f : \mathcal{X} \to \mathcal{Y}$. When sampling a SWIM-sampled network as defined in Definition A.2, for hidden layers $l = 1, \ldots, L$ the probability density p_l^ϵ to sample pairs of points to be used for the corresponding hidden layer can be defined through the proportionality

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

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 sub-network $\Phi^{(l-1)}$ parametrized 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 image after $(l-1)$ layers,
- assuming distinct inputs to the network we have $\epsilon = 0$ for $l = 1$ and $\epsilon > 0$ otherwise. We only experiment with shallow networks in this paper so we do not have to set this parameter in our experiments,
- the 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 our experiments.

The probability density above can only be used in a supervised setting. The networks that we construct that use this distribution using the true function values $f(\cdot)$ are denoted as **SWIM**, and the ones that use this distribution using approximate function values are denoted as **A-SWIM**. For the initial approximation, we use the uniform distribution over the pairs of points in the input space $\mathcal{X}$. The networks that are constructed using the uniform distribution are denoted as **U-SWIM**.

Algorithm A.1: Our proposed HNN sampling algorithm is illustrated. $\mathcal{L}$ is a loss function, which in our case is the L^2 loss, and $\arg\min \mathcal{L}(\cdot, \cdot)$ becomes a linear optimization problem that we solve using the least squares solution $\arg\min_w \|Aw - u\|^2$ of the linear system in Equation (3).

Data: $\{x_i, \dot{x}_i\}_{i=1}^K$, $\{x_0, \mathcal{H}(x_0)\}$, *sampler* $\in \{\texttt{ELM}, \texttt{U-SWIM}, \texttt{A-SWIM}\}$

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

B Numerical experiments

The experiments in this section are conducted 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 $\approx$ 96 GB memory (bandwidth 80 GB/s). Note that we have only used a single node with 64 threads in all the experiments.

For a fair evaluation, we train the networks and take the mean error values of 10 randomly conducted runs, which include different seeds for the model and the generated distinct train and test sets. We use the linear solver `numpy.linalg.lstsq` for solving the least-squares optimization problem with python and numpy [16] versions 3.12.4 and 1.26.4, respectively. We train the HNNs in our experiments with the HNN loss [12] using the Adam optimizer implementation `torch.optim.Adam` from torch [27] version 2.4.0. The weights of the HNNs are initialized using the Xavier normal distribution [11], and the biases of the HNNs are zero-initialized. Similarly to the sampled HNNs, we set the integration constant, i.e., the last layer bias, accordingly assuming we know the true function value for a single data point. We used double precision (float64, both in numpy and pytorch) in our experiments. For more information regarding other software versions, random seeds, and the implementation, please refer to the open-source code repository available at `https://github.com/AlphaGergedan/Sampling-HNNs`.

In all the figures, we plot the mean error values of multiple runs and paint the area around this error curve using the minimum and maximum error values to demonstrate the consistency of the methods using the Matplotlib library [20].

In all the experiments, we used SWIM sampling with the approximate values resulting from U-SWIM for the A-SWIM method. For comparison, we also used SWIM sampling with the true function values and denoted this method as SWIM in the figures.

Error function: We compare the relative (rel.) L^2 error

$$\sqrt{\frac{\sum_i \left(\mathcal{H}(q_i,p_i) - \widehat{\mathcal{H}}(q_i,p_i)\right)^2}{\sum_i \mathcal{H}(q_i,p_i)^2}}$$

when analyzing and evaluating the approximations using all the points in the test set, where $\widehat{\mathcal{H}}$ represents the final approximation (trained model) and $\mathcal{H}$ the true Hamiltonian. In all tables, the mean error value of 10 randomly conducted runs is given for each entry for sampled networks.

Target Hamiltonian functions: Target Hamiltonian functions, which we aim to approximate in this work, are the following. The single pendulum Hamiltonian

$$\begin{aligned}\mathcal{H}(q,p) &= \frac{p^2}{2ml^2} + mgl(1-\cos(q)) \\ &= \frac{p^2}{2} + (1-\cos(q))\end{aligned} \tag{B.4}$$

for the experiments in Figure B.3, where we set the constants (mass m, link length l, gravitational acceleration g) to 1. The single pendulum is extended with a frequency parameter resulting in the function

$$\mathcal{H}(q,p) = \frac{p^2}{2} + (1-\cos(fq)) \tag{B.5}$$

for the experiments in Figure 2. The Lotka-Volterra [10] Hamiltonian

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

with parameters $\beta=-1,\alpha=-2,\delta=-1,\gamma=-1$ and $\beta=0.025,\alpha=3.5,\delta=0.07,\gamma=10$ for the zero-centered and five-centered domains, respectively, resulting in the functions

$$\mathcal{H}(q,p) = -e^q + 2q - e^p + p \tag{B.6}$$

and

$$\mathcal{H}(q,p) = 0.025e^q - 3.5q + 0.07e^p - 10p \tag{B.7}$$

for the experiments in Figure B.4. The double pendulum Hamiltonian

$$\begin{aligned}\mathcal{H}(q,p) &= \frac{m_2 l_2^2 p_1^2 + (m_1+m_2) l_1^2 p_2^2 - 2m_2 l_1 l_2 p_1 p_2 \cos(q_1-q_2)}{2m_2 l_1^2 l_2^2 (m_1 + m_2 \sin^2(q_1-q_2))} \\ &\quad - (m_1+m_2) g l_1 \cos(q_1) - m_2 g l_2 \cos(q_2) \\ &= \frac{p_1^2 + 2p_2^2 - 2p_1 p_2 \cos(q_1-q_2)}{2(1+\sin^2(q_1-q_2))} - 2\cos(q_1) - \cos(q_2),\end{aligned} \tag{B.8}$$

where we set the constants (mass m_1 and m_2, link lengths l_1 and l_2, gravitational acceleration g) to 1 for the experiments in Table 2. The Hénon-Heiles [17] Hamiltonian

$$\mathcal{H}(q,p) = \frac{1}{2}(p_1^2 + p_2^2) + \frac{1}{2}(q_1^2 + q_2^2) + \alpha(q_1^2 q_2 - \frac{1}{3}q_2^3), \tag{B.9}$$

where we set the bifurcation parameter $\alpha = 1$ for the experiments in Table 2.

Single pendulum: We show that all the methods can reach very low approximation errors in Figure B.3 when approximating the single pendulum Hamiltonian.

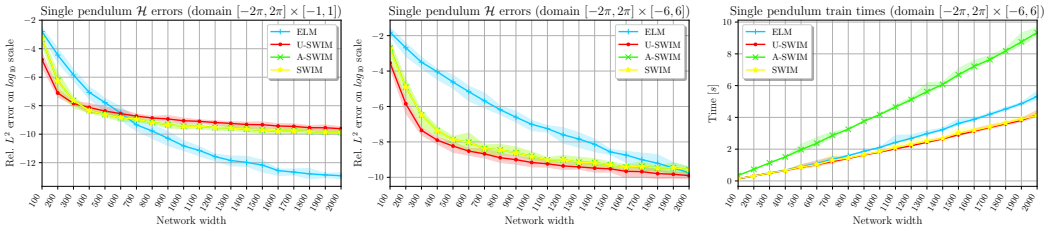

Figure B.3: Single pendulum approximation errors and training times for the larger domain are plotted. See Table B.5 and Table B.3 for domain and model parameters.

Especially in the larger domain and with smaller network widths, SWIM sampling outperformed ELM. On the other hand, while the SWIM methods' performance remained consistent, ELM could reach lower approximation errors in both domains with large network widths (more than 600 in the smaller, 2000 in the larger domain). In the larger domain, the gradients of the system move away from zero, and with large gradients where the function values change quickly in the target function, the SWIM methods might have constructed better (more accurate and consistent) weights using the data-driven scheme with small network widths. Also, ELM may need larger network widths to be able to cover the input space uniformly in larger domains. Remarkably, A-SWIM can match the SWIM method's accuracy in all the settings with an additional training time cost of resampling after the initial approximation. 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 SWIM requires the true function values.

Lotka-Volterra: We experiment with Lotka-Volterra Hamiltonians in different settings in Figure B.4. Similar to the single pendulum experiments, the ELM method performed better with large network widths than the SWIM sampling, however, its performance has declined in the larger domains where the gradients are larger. Also, note that when the equilibrium changes to a vector close to 5 in both dimensions, ELM method becomes less consistent.

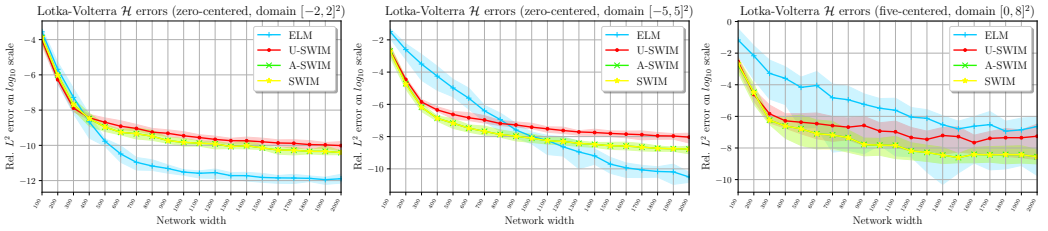

Figure B.4: Lotka-Volterra approximation errors are plotted. The target Hamiltonian in the left and center plots has an equilibrium near the zero-vector, whereas the target Hamiltonian in the right plot has an equilibrium around five in both dimensions. Domain and model parameters are set according to Table B.5 and Table B.3

Chaotic-systems: In addition to the results in Table 2, we provide network scaling for the double pendulum and note that all the methods perform similarly in Figure B.5. To efficiently train HNNs using backpropagation, we have also tried using the NVIDIA GeForce RTX 3060 Mobile / Max-Q graphics card with CUDA version 12.4 for the chaotic systems using the same settings as in Table 2. GPU time of the double pendulum and Hénon-Heiles experiment was 7313.1 and 7788.9 seconds respectively. Using single precision has resulted in ≈ 2.5 times faster training without losing a lot of accuracy. However, training HNNs using our approach was still more than 20 times faster with CPUs.

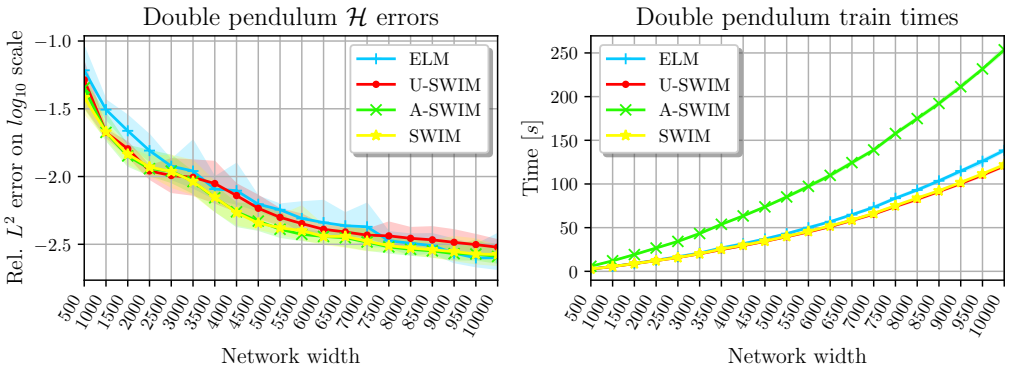

Figure B.5: Double pendulum approximation errors and training times are displayed. Network width was scaled and other model parameters were set as described in Table B.4. Domain information is listed in Table B.5.

Table B.3 and Table B.4 list model parameters, and Table B.5 list domain parameters used in the numerical experiments in this work. ELM bias distribution specifies the distribution used for sampling the hidden layer biases for the ELM method. For all the experiments we use the `Uniform` distribution with $\min$ and $\max$ set to the minimum and maximum domain range. Note that we do not fix this distribution to have a more fair comparison of the weight sampling used in ELM compared to other methods, as we have noticed a strong decline in the accuracy of ELM if the bias is sampled from a fixed range and the target domain is scaled drastically. For the SWIM methods, we make sure that different pairs of points are selected for different neurons for the weight construction to avoid any duplicates by re-sampling if a duplicate is detected and repeat this until we get unique pairs, i.e., unique parameters.

Conservation of the Hamiltonian value: Greydanus et al. [12] have demonstrated energy conservation of traditional HNNs compared to plain MLPs (which directly output system dynamics $\dot{q}$ and $\dot{p}$) along trajectory predictions. We demonstrate this property using sampled networks too in Figure B.6.

Learning from trajectory data with error correction: Learning from trajectory data consisting of generalized "position" q and "momentum" p coordinates only has been studied in many recent work. Zhu et al. [37] theoretically analyzes the training of traditional HNNs using finite differences with symplectic and non-symplectic integration schemes. Networks trained with integrator schemes learn the network targets rather than the true Hamiltonian due to discretization errors. Their work aligns with the numerical results presented by Chen et al. [4], where the trained networks account for the numerical errors. An adapted loss was proposed by Zhu et al. [37], which is then adapted for Gaussian Processes by Offen and Ober-Bloebaum [25]. In later work, David and Mehats [6] adapted a post-training correction method to account for the discretization errors for traditional HNNs trained with symplectic integrators to recover the true Hamiltonian function of the target system. In the rest of this section, we incorporate this correction scheme into Equation (3) and demonstrate that it can also be applied to sampled HNNs that we introduce in this paper.

Given a trajectory dataset $\mathcal{D} = \{x_i, \varphi_h(x_i)\}_{i=1}^K$ where φ_h is the exact flow of the target Hamiltonian system with time step size $\Delta t = h > 0$. Note that any trajectory of arbitrary length can be written to

Table B.3: Model parameters in the single pendulum (Figure B.3 and Table 1), single pendulum with frequency (Figure 2), and Lotka-Volterra (Figure B.4 and Table 1) experiments are listed. ELM bias distribution is set using the minimum and maximum domain boundaries. Therefore, for Lotka-Volterra experiments, it is set as $\texttt{Uniform}(\{-2,-5,0\},\{2,5,8\})$ depending on the domain. The last four columns are the parameters of the traditionally trained HNN, including the total number of gradient steps in training, learning rate and weight decay in the Adam optimizer, and batch size. For Lotka-Volterra experiments, the total number of gradient steps is set to either 15000 or 30000, depending on the domain size.

Parameter	Single pendulum	Single pendulum with f	Lotka-Volterra
Number of layers	1	1	1
Network width	See fig. B.3	1500	See fig. B.4
Activation	$\tanh$	$\tanh$	$\tanh$
L^2 regularization	10^{-13}	10^{-13}	10^{-13}
ELM bias distribution	$\texttt{Uniform}(-\pi,\pi)$	$\texttt{Uniform}(-\pi,\pi)$	See table caption
HNN #gradient-steps	15000	-	See table caption
HNN learning rate	$5\cdot10^{-4}$	-	$5\cdot10^{-4}$
HNN weight decay	10^{-13}	-	10^{-13}
HNN batch size	2048	-	2048

Table B.4: Model parameters in the double pendulum and Hénon-Heiles experiments (see Table 2) are listed.

Parameter	Double pendulum	Hénon-Heiles
Number of layers	1	1
Network width	5000	5000
Activation	$\tanh$	$\tanh$
L^2 regularization	10^{-13}	10^{-13}
ELM bias distribution	$\texttt{Uniform}(-\pi,\pi)$	$\texttt{Uniform}(-5,5)$
HNN #gradient-steps	180000	180000
HNN learning rate	10^{-4}	10^{-4}
HNN weight decay	10^{-13}	10^{-13}
HNN batch size	2048	2048

Table B.5: Domain parameters of the target Hamiltonians in the experiments are listed.

System name	Hamiltonian	Domain	Train set size	Test set size
Single pendulum	eq. (B.4)	$[-2\pi,2\pi]\times[-1,1]$	10000	10000
Single pendulum	eq. (B.4)	$[-2\pi,2\pi]\times[-6,6]$	10000	10000
Single pendulum	eq. (B.5)	$[-\pi,\pi]\times[-0.5,0.5]$	See fig. 2	10000
Lotka-Volterra	eq. (B.6)	$[-2,2]\times[-2,2]$	10000	10000
Lotka-Volterra	eq. (B.6)	$[-5,5]\times[-5,5]$	10000	10000
Lotka-Volterra	eq. (B.7)	$[0,8]\times[0,8]$	10000	10000
Double pendulum	eq. (B.8)	$[-\pi,\pi]^2\times[-1,1]^2$	20000	20000
Hénon-Heiles	eq. (B.9)	$[-5,5]^2\times[-5,5]^2$	20000	20000

have the form of $\mathcal{D}$ if the points in the trajectory are considered as pairs. Similar to [25], we adapt the linear system Equation (3), which we aim to solve, to use the finite differences on the right-hand side, and the symplectic Euler integration scheme on the left-hand side:

$$\begin{bmatrix} \nabla\Phi^{(L)}(\varphi_h(q_1),p_1) & 0 \\ \nabla\Phi^{(L)}(\varphi_h(q_2),p_2) & 0 \\ \vdots & \vdots \\ \nabla\Phi^{(L)}(\varphi_h(q_K),p_K) & 0 \\ \Phi^{(L)}(x_0) & 1 \end{bmatrix} \cdot \begin{bmatrix} W_{L+1}^{\mathsf{T}} \\ b_{L+1} \end{bmatrix} \overset{!}{=} \begin{bmatrix} J^{-1}\cdot\left((\varphi_h(x_1)-x_1)/h\right) \\ J^{-1}\cdot\left((\varphi_h(x_2)-x_2)/h\right) \\ \vdots \\ J^{-1}\cdot\left((\varphi_h(x_K)-x_K)/h\right) \\ \mathcal{H}(x_0) \end{bmatrix}. \tag{B.10}$$

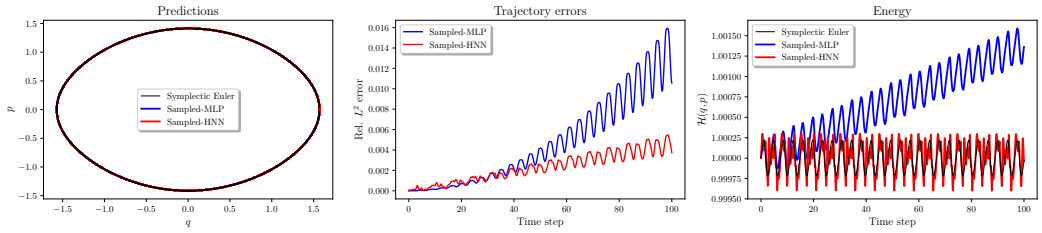

Figure B.6: Sampled-HNN outputs the Hamiltonian value, while Sampled-MLP directly approximates the time derivatives of the inputs $\dot{q}$ and $\dot{p}$. Single pendulum is approximated with the same domain and model parameters used in the larger domain single pendulum experiments (domain $[-2\pi, 2\pi] \times [-6, 6]$) and both Sampled-MLP and Sampled-HNN are sampled according to the ELM method. The trained models and the true Hamiltonian are integrated using symplectic Euler with time step size $5 \cdot 10^{-4}$ from the initial coordinates $x_0 = (\pi/2, 0)$.

David and Mehats [6] used the well-established error analysis by Hairer et al. [13] (Chapter 9, Example 3.4) for the symplectic Euler method

$$\widehat{\mathcal{H}} = \mathcal{H} - \frac{h}{2}\nabla_q \mathcal{H}^\mathsf{T} \nabla_p \mathcal{H} + \mathcal{O}(h^2),$$

and corrected the discretization error of a trained traditional HNN up to an order as

$$\mathcal{H} \approx \widehat{\mathcal{H}} + \frac{h}{2}\nabla_q \widehat{\mathcal{H}}^\mathsf{T} \nabla_p \widehat{\mathcal{H}}.$$

We use the same correction scheme using analytical solutions for a sampled HNN Φ as

$$\mathcal{H} \approx \Phi + \frac{h}{2}\nabla_q \Phi^\mathsf{T} \nabla_p \Phi,$$

and demonstrate the error correction for the single pendulum system on domain $[-\pi, \pi] \times [-1, 1]$ with train set size 2500 with their next states after time step h, test set size 2500, L^2 regularization 10^{-13} and network width 200 using the A-SWIM method for the sampling in Figure B.7. The true flow of the target system is simulated using the explicit Runge-Kutta method of order 5(4) implementation from `scipy.integrate.solve_ivp` [31] with time step size 10^{-4}.

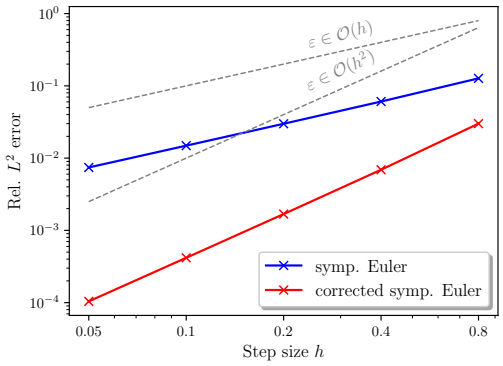

Figure B.7: Single pendulum Hamiltonian approximation errors are displayed with and without error correction together with reference lines $\epsilon = h$ and $\epsilon = h^2$. Symp. Euler stands for the A-SWIM method using finite differences and symplectic Euler integration scheme as described in Equation (B.10).