LEARNING THE DYNAMICS OF PHYSICAL SYSTEMS WITH HAMILTONIAN GRAPH NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Inductive biases in the form of conservation laws have been shown to provide superior performance for modeling physical systems. Here, we present Hamiltonian graph neural network (HGNN), a physics-informed GNN that learns the dynamics directly from the trajectory. We evaluate the performance of HGNN on spring, pendulum, and gravitational systems and show that it outperforms other Hamiltonian-based neural networks. We also demonstrate the zero-shot generalizability of HGNN to unseen hybrid spring-pendulum systems and system sizes that are two orders of magnitude larger than the training systems. HGNN provides excellent inference in all the systems providing a stable trajectory. Altogether, HGNN presents a promising approach to modeling complex physical systems directly from their trajectory.

1 INTRODUCTION AND RELATED WORKS

Learning the dynamics of physical systems directly from their trajectory is a problem of interest in wide areas such as robotics, mechanics, biological systems such as proteins, and atomistic dynamics (Cranmer et al., 2020b; Karniadakis et al., 2021). Recently, it has been shown that infusing the inductive biases in terms of the ODEs directly in the formulation as a hard constraint as in Hamiltonian (HNN) (Sanchez-Gonzalez et al., 2019; Greydanus et al., 2019; Zhong et al., 2020; 2021), and Lagrangian neural networks (LNN) (Cranmer et al., 2020a; Finzi et al., 2020; Lutter et al., 2019)—instead of a soft constraint in the loss function—can significantly enhance the learning efficiency while also leading to realistic trajectories in terms of conservation laws. Additionally, combining these formulations with graph neural networks (GNNs) (Scarselli et al., 2008; Bhattoo et al., 2023; 2022; Thangamuthu et al., 2022) can lead to superior properties such as zero-shot generalizability to unseen system sizes and hybrid systems unseen during the training, more efficient learning, and inference.

Of special interest among these physics-informed GNNs are the Hamiltonian graph neural networks (HGNN), where you employ Hamiltonian equations of motion as an inductive bias. In the present work, we propose a HGNN that learns the Hamiltonian of a system directly from the trajectory. We evaluate our architecture on several complex systems such as $n-$ pendulum and spring systems and gravitational systems. The major contribution of our work is as follows.

- **Graph architecture:** We propose a graph architecture for HGNN which decouples body forces and internal forces, which allows efficient learning for systems with external forces such as $n-$ pendulums.
- **Decoupling kinetic and potential energies:** Our HGNN architecture decouples the potential and kinetic energies, which, in turn, allows improved interpretation of the learned quantities.
- **Explicit constraints:** We show that employing explicit constraints allows efficient learning in systems such as pendulums where such constraints govern the dynamics of the systems.
- **Zero-shot generalizability:** Finally, we show that HGNN exhibits generalizability to system sizes that are orders of magnitude larger than the training trajectory and to hybrid systems. For instance, HGNN trained on spring and pendulum systems can infer the dynamics of a complex spring-pendulum system.

2 BACKGROUND ON HAMILTONIAN MECHANICS

Consider a system of n particles that are interacting with their positions at time t represented by the Cartesian coordinates as $\mathbf{x}(t) = (\mathbf{x_1}(t), \mathbf{x_2}(t), ...\mathbf{x_n}(t))$. The Hamiltonian H of the system is defined

as $H(\mathbf{p_x}, \mathbf{x}) = T(\dot{\mathbf{x}}) + V(\mathbf{x})$, where $T(\dot{x})$ represents the total kinetic energy and $V(\mathbf{x})$ represents the potential energy of the system. The Hamiltonian equations of motion for this system in Cartesian coordinates are given by (LaValle, 2006; Goldstein, 2011)

$$\dot{\mathbf{x}} = \nabla_{\mathbf{p_x}} H, \qquad \dot{\mathbf{p_x}} = -\nabla_{\mathbf{x}} H \tag{1}$$

where $\mathbf{p_x} = \nabla_{\dot{x}} H = \mathbf{M}\dot{\mathbf{x}}$ represents the momentum of the system in Cartesian coordinates and $\mathbf{M}$ represents the mass matrix. Assuming $Z = [\mathbf{x}; \mathbf{p_x}]$ and $J = [0, I; -I, 0]$, Hamiltonian equations can be combined as

$$\nabla_Z H + J\dot{Z} = 0 \tag{2}$$

In systems with constraints, as in the case of a pendulum, the Hamiltonian equations of motion can be modified to feature the constraints explicitly as

$$\nabla_Z H + J\dot{Z} + (D_Z \Psi)^T \lambda = 0 \tag{3}$$

where $(D_Z \Psi)^T \lambda$ represents the effect of constraints on $\dot{\mathbf{x}}$ and $\dot{\mathbf{p_x}}$. Here, $\Psi(Z) = (\Phi; \dot{\Phi})$ and $\Phi = \Phi(\mathbf{x}) = 0$ represent the constraints equation $\Psi(Z) = 0$. Thus, $(D_Z \Psi)\dot{Z} = 0$. Substituting for $\dot{Z}$ and solving for λ yields

$$\lambda = -[(D_Z \Psi) J (D_Z \Psi)^T]^{-1} [(D_Z \Psi) J (\nabla H)] \tag{4}$$

Substituting λ in the Eq. 3 and solving for $\dot{Z}$ yields

$$\dot{Z} = J[\nabla_Z H - (D_Z \Psi)^T [(D_Z \Psi) J (D_Z \Psi)^T]^{-1} (D_Z \Psi) J \nabla_Z H] \tag{5}$$

In this work, we employ Eq.5 to obtain the acceleration of the system from the Hamiltonian, which, when integrated, provides the updated configuration of the system.

3 Graph architecture

In this section, we briefly describe the architecture of HGNN shown in Fig. 1.

Graph structure. The physical system is modeled as an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with nodes as particles and edges as connections between them. For instance, in a n-ball-spring system, the balls are represented as nodes and springs as edges.

Input features. The raw node features are t (type of particle) as one-hot encoding, $\mathbf{x}$, and $\dot{\mathbf{x}}$, and the raw edge feature is the distance, $d = ||\mathbf{x}_j - \mathbf{x}_i||$, between two particles i and j. A notable difference in the HGNN architecture from previous works is the presence of global and local features—local features participate in message passing and contribute to quantities that depend on topology, while global features do not take part in message passing. Here, we employ the position $\mathbf{x}$, velocity $\dot{\mathbf{x}}$ as global features for a node while d and t are used as local features.

Neural architecture. We employ a l-layer message passing GNN, which takes an embedding of the node and edge features created by MLPs as input. The local features participate in message passing to create an updated embedding for both the nodes and edges. The final representations of the nodes and edges, z_i and z_{ij}, respectively, are passed through MLPs to obtain the Hamiltonian of the system.

Hamiltonian prediction. The Hamiltonian of the system is predicted as the sum of T and V in the HGNN. Specifically, the potential energy is predicted as $V_i = \sum_i MLP_v(z_i) + \sum_{ij} MLP_e(z_{ij}))$, where MLP_v and MLP_e represent the contribution from the node (particles themselves) and edges (interactions) toward the potential energy of the system, respectively, and $T = \sum_i MLP_T(h_i^T)$.

Trajectory prediction and training. The H of the system is obtained from HGNN is substituted in the Eq.(3) to obtain the acceleration and velocity of the particles. These values are integrated using velocity Verlet, a symplectic integrator, to compute the updated position. The loss function of HGNN is computed by using the predicted and actual positions at timesteps $2, 3, \ldots, \mathcal{T}$ in a trajectory $\mathbb{T}$, which is then back-propagated to train the MLPs. Specifically, the loss function is as follows.

$$\mathcal{L} = \frac{1}{n} \left(\sum_{i=1}^{n} \sum_{t=2}^{\mathcal{T}} \left(\mathbf{x}_i^{\mathbb{T},t} - \hat{\mathbf{x}}_i^{\mathbb{T},t} \right)^2 \right) \tag{6}$$

4 Empirical evaluations

In this section, we evaluate the performance of HGNN toward (i) learning the dynamics of physical systems, (ii) generalizing to unseen hybrid systems, and (iii) generalizing to unseen system sizes. Our implementation is available at `https://anonymous.4open.science/r/HGNN-37C6`.

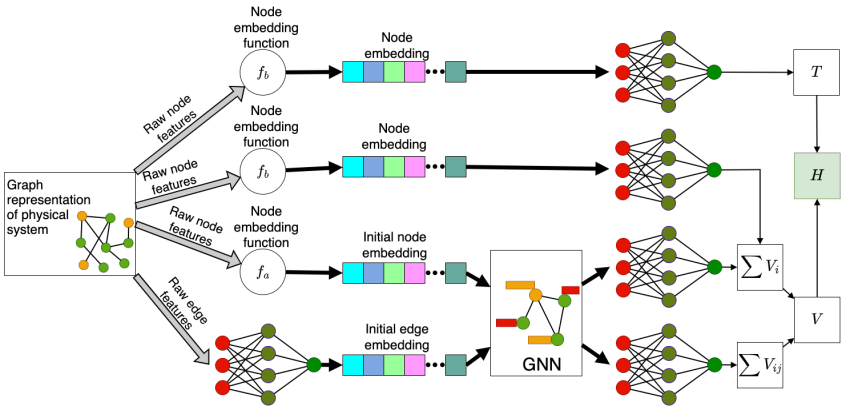

Figure 1: Architecture of Hamiltonian Graph Neural Network

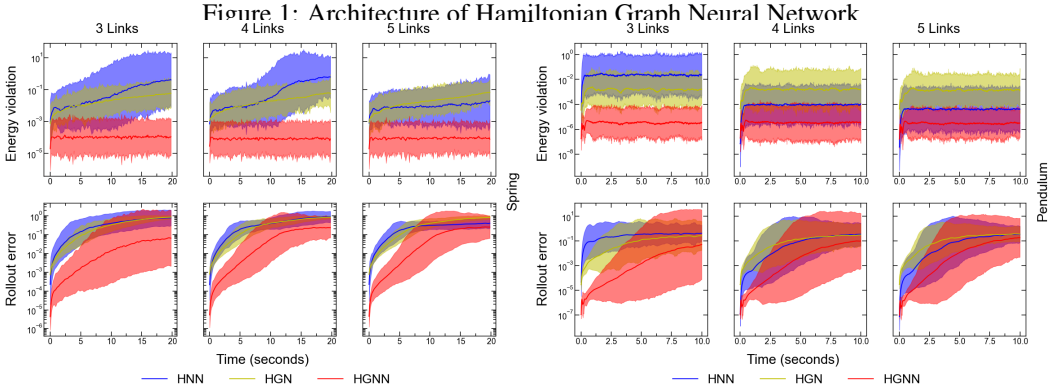

Figure 2: EE and RE for 5-,10-,50- links pendulum and 5-,50-,500- links spring systems.

Details of the systems are provided in the App. B.

Baselines: We consider two baselines to compare the performance of HGNN. The first, HNN (Greydanus et al., 2019), is a simple MLP that directly predicts the Hamiltonian of the system. Note that the decoupling of kinetic and potential energies is implemented in HNN. Second, HGN (Sanchez-Gonzalez et al., 2019) is a graph-based version of HNN, albeit without decoupling the kinetic and potential energies. While the performance of HNN has been demonstrated on several spring and pendulum systems, HGN (Sanchez-Gonzalez et al., 2019) has been evaluated only on spring systems.

• **Datasets and systems:** To evaluate HGNN, we selected standard systems, *viz*, n-pendulums and springs, where $n = (3, 4, 5)$. All the graph-based models are trained on 5-pendulum and 5-spring systems only, which are then evaluated on other system sizes. Further, to evaluate the zero-shot generalizability of HGNN to large-scale unseen systems, we simulate 5, 50, 500-link spring systems, and 5-, 10-, and 50-link pendulum systems. We also considered a hybrid spring-pendulum system unseen during training to evaluate HGNN and a gravitational system. The detailed data-generation procedure is given in App. C.1. The timestep used for the forward simulation of the pendulum system is $10^{-5}s$ with the data collected every 1000 timesteps, and for the spring system is $10^{-3}s$ with the data collected every 100 timesteps. Model architecture and training details are provided in the App. A.

• **Evaluation Metric:** Following the work of (Finzi et al., 2020), we evaluate performance by computing the following three error metrics, namely, (1) relative error in the trajectory, defined as the *rollout error, (RE(t))*, (2) *Energy violation error (EE(t))* given by $RE(t) = \frac{||\hat{q}(t)-q(t)||_2}{||\hat{q}(t)||_2+||q(t)||_2}$, $EE(t) = \frac{||\hat{\mathcal{H}}-\mathcal{H}||_2}{(||\hat{\mathcal{H}}||_2+||\mathcal{H}||_2}$. Note that all the variables with a hat, for example $\hat{x}$, represent the predicted values based on the trained model and the variables without a hat, that is x, represent the ground truth.

4.1 Rollout and Energy Errors

Fig. 2 shows the performance of HNN, HGN, and HGNN for spring and pendulum systems. We observe that HGNN outperforms both HNN and HGN on both spring and pendulum systems. Specifically, we

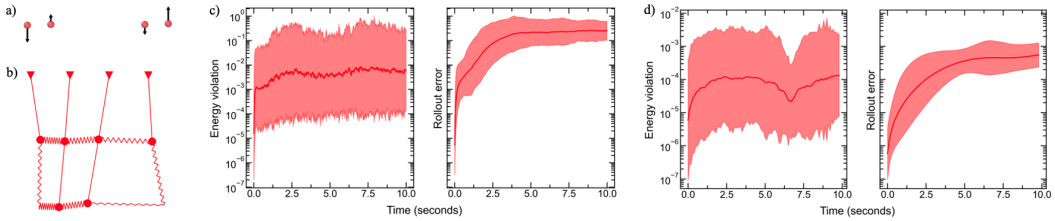

Figure 3: Visualization of (a) gravitational and (b) hybrid systems. EE and RE for (c) hybrid system and (d) 4-body gravitational system.

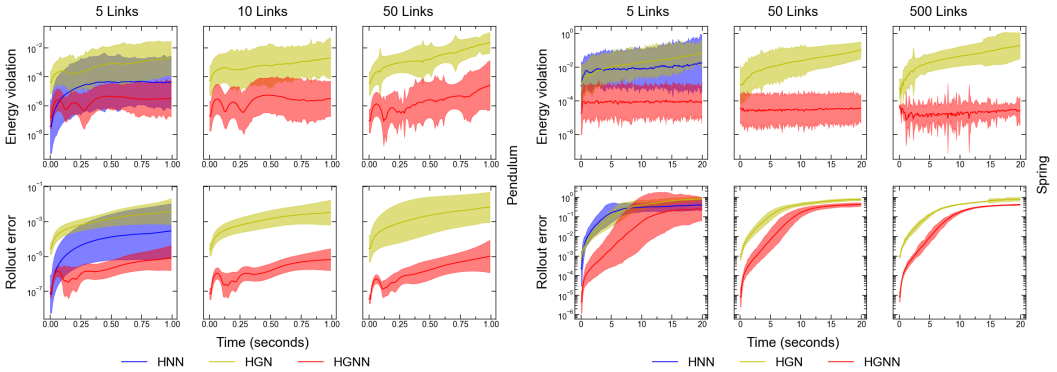

Figure 4: Energy violation and Rollout error for 5-,10-,50-links pendulum and 5-,50-,500-links spring systems.

observe that the energy violation error in HGNN remains saturated, suggesting a stable and realistic predicted trajectory. Note that HNN is trained and evaluated on each of these systems separately, while HGN and HGNN are trained in only one system and inferred for all other systems by performing the forward simulation.

4.2 Complex systems

In order to evaluate the performance of HGNN on more complex systems, we consider a gravitation system and a hybrid spring-pendulum system (see Figs. 3(a) and (b)). We observe that HGNN, trained on spring and pendulum systems separately, provides an excellent inference for the hybrid system unseen by the model. Despite best efforts, the HGN and HNN was unable to provide a forward trajectory for the hybrid system. The superior performance of HGNN could be attributed to the architecture, which decouples the potential and kinetic energies and learns them separately for each system. We also evaluate HGNN for a more complex interaction than springs and pendulums, that is, gravitational forces. Fig. 3 shows that HGNN provides excellent inference for the gravitational system. Similar to the hybrid system, the baselines trained on the gravitational systems were unable to provide a stable trajectory and exploded after a few steps during the inference.

4.3 Zero-shot generalization

Finally, we evaluate the zero-shot generalizability of HGNN in comparison to HGN (see Fig. 4). We observe that HGNN exhibits superior generalization to system sizes that are two orders of magnitude larger than the training system. In the case of the spring system, even for a system size two orders of magnitude error, we observe a comparable error in energy, which remains stable with time.

5 Conclusion

In this work, we present a HGNN that learns the Hamiltonian of physical systems directly from their trajectory. We demonstrate the approach to spring, pendulum, and gravitational systems. HGNN outperforms HNN and HGN both in terms of energy and rollout errors. Further, HGNN exhibits zero-shot generalizability to unseen hybrid systems and large-scale systems that are two orders of magnitude larger than the training system. While HGNN presents a promising approach, it is still applied to simple systems. Extending this to more complex systems, such as atomic structures or deformable bodies, can be addressed as future challenges.

REFERENCES

Ravinder Bhattoo, Sayan Ranu, and NM Anoop Krishnan. Learning articulated rigid body dynamics with lagrangian graph neural network. In *Advances in Neural Information Processing Systems*, 2022.

Ravinder Bhattoo, Sayan Ranu, and NM Anoop Krishnan. Learning the dynamics of particle-based systems with lagrangian graph neural networks. *Machine Learning: Science and Technology*, 2023.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, and Skye Wanderman-Milne. Jax: composable transformations of python+ numpy programs, 2018. *URL http://github. com/google/jax*, 4:16, 2020.

Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian neural networks. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2020a. URL `https://openreview.net/forum?id=iE8tFa4Nq`.

Miles Cranmer, Alvaro Sanchez Gonzalez, Peter Battaglia, Rui Xu, Kyle Cranmer, David Spergel, and Shirley Ho. Discovering symbolic models from deep learning with inductive biases. *Advances in Neural Information Processing Systems*, 33, 2020b.

Marc Finzi, Ke Alexander Wang, and Andrew G Wilson. Simplifying hamiltonian and lagrangian neural networks via explicit constraints. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 13880–13889. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/9f655cc8884fda7ad6d8a6fb15cc001e-Paper.pdf`.

Jonathan Godwin*, Thomas Keck*, Peter Battaglia, Victor Bapst, Thomas Kipf, Yujia Li, Kimberly Stachenfeld, Petar Veličković, and Alvaro Sanchez-Gonzalez. Jraph: A library for graph neural networks in jax., 2020. URL `http://github.com/deepmind/jraph`.

Herbert Goldstein. *Classical mechanics*. Pearson Education India, 2011.

Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. *Advances in Neural Information Processing Systems*, 32:15379–15389, 2019.

George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

Michael Lutter, Christian Ritter, and Jan Peters. Deep lagrangian networks: Using physics as model prior for deep learning. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=BklHpjCqKm`.

Alvaro Sanchez-Gonzalez, Victor Bapst, Kyle Cranmer, and Peter Battaglia. Hamiltonian graph networks with ode integrators. *arXiv e-prints*, pp. arXiv–1909, 2019.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

Samuel Schoenholz and Ekin Dogus Cubuk. Jax md: a framework for differentiable physics. *Advances in Neural Information Processing Systems*, 33, 2020.

Abishek Thangamuthu, Gunjan Kumar, Suresh Bishnoi, Ravinder Bhattoo, NM Anoop Krishnan, and Sayan Ranu. Unravelling the performance of physics-informed graph neural networks for dynamical systems. In *Advances in Neural Information Processing Systems*, 2022.

Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Dissipative symoden: Encoding hamiltonian dynamics with dissipation and control into deep learning. *arXiv preprint arXiv:2002.08860*, 2020.

Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Benchmarking energy-conserving neural networks for learning dynamics from data. In *Learning for Dynamics and Control*, pp. 1218–1229. PMLR, 2021.

A MODEL ARCHITECTURE AND TRAINING SETUP:

For HGN, HNN, and HGNN, the MLPs are two-layers deep. We use 10000 data points from 100 trajectories divided into 75:25 (train: validation) to train all the models. Detailed training procedures and hyper-parameters are provided in App. C.2. All models were trained till the decrease in loss saturates to less than 0.001 over 100 epochs. The model performance is evaluated on a forward trajectory, a task it was not explicitly trained for, of $10s$ in the case of the pendulum and $20s$ in the case of spring. Note that this trajectory is 2-3 orders of magnitude larger than the training trajectories from which the data has been sampled. The dynamics of n-body system are known to be chaotic for $n \geq 2$. Hence, all the results are averaged over trajectories generated from 100 different initial conditions.

A.1 SIMULATION ENVIRONMENT

All the simulations and training were carried out in the JAX environment (Schoenholz & Cubuk, 2020; Bradbury et al., 2020). The graph architecture was developed using the jraph package (Godwin* et al., 2020). The experiments were conducted on a machine with Apple M1 chip having 8GB RAM and running MacOS Monterey.

B EXPERIMENTAL SYSTEMS

To simulate the ground truth, physics-based equations derived using Lagrangian mechanics is employed. The equations for n-pendulum and spring systems are given in detail below.

B.1 n-PENDULUM

For an n-pendulum system, n-point masses, representing the bobs, are connected by rigid (non-deformable) bars. These bars, thus, impose a distance constraint between two point masses as

$$||x_i - x_{i-1}||^2 = l_i^2 \tag{7}$$

where, l_i represents the length of the bar connecting the $(i-1)^{th}$ and i^{th} mass. This constraint can be differentiated to write in the form of a *Pfaffian* constraint as

$$(x_i - x_{i-1})(\dot{x}_i - \dot{x}_{i-1}) = 0 \tag{8}$$

Note that such constraint can be obtained for each of the n masses considered to obtain the $A(q)$.

The Lagrangian of this system can be written as

$$L = \sum_{i=1}^{n} \left(1/2 m_i \dot{x_i}^{\mathrm{T}} \dot{x_i} - m_i g x_i^{(2)} \right) \tag{9}$$

where m_i represents the mass of the ith particle, g represents the acceleration due to gravity in the $x^{(2)}$ direction and $x^{(2)}$ represents the position of the particle in the $x^{(2)}$ direction.

B.2 n-SPRING SYSTEM

Here, n-point masses are connected by elastic springs that deform linearly (elastically) with extension or compression. Note that similar to the pendulum setup, each mass m_i is connected to two masses m_{i-1} and m_{i+1} through springs so that all the masses form a closed connection. The Lagrangian of this system is given by

$$L = \sum_{i=1}^{n} 1/2 m_i \dot{x_i}^{\mathrm{T}} \dot{x_i} - \sum_{i=1}^{n} 1/2 k (||x_{i-1} - x_i|| - r_0)^2 \tag{10}$$

where r_0 and k represent the undeformed length and the stiffness, respectively, of the spring.

B.3 n-BODY GRAVITATIONAL SYSTEM

Here, n point masses are in a gravitational field generated by the point masses themselves. The Lagrangian of this system is given by

$$L = \sum_{i=1}^{n} 1/2 m_i \dot{x_i}^{\mathrm{T}} \dot{x_i} + \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} G m_i m_j / 2(||x_i - x_j||) \tag{11}$$

where G represents the Gravitational constant.

C IMPLEMENTATION DETAILS

C.1 DATASET GENERATION

Software packages: numpy-1.22.1, jax-0.3.0, jax-md-0.1.20, jaxlib-0.3.0, jraph-0.0.2.dev
Hardware: Chip: Apple M1, Total Number of Cores: 8 (4 performance and 4 efficiency), Memory: 8 GB, System Firmware Version: 7459.101.3, OS Loader Version: 7459.101.3

C.2 HYPER-PARAMETERS

The default hyper-parameters used for training each architecture is provided below.

•HNN

Parameter	Value
Hidden layer neurons (MLP)	256
Number of hidden layers (MLP)	2
Activation function	squareplus
Optimizer	ADAM
Learning rate	$1.0e^{-3}$
Batch size	100

•HGN

Parameter	Value
Node embedding dimension	8
Edge embedding dimension	8
Hidden layer neurons (MLP)	16
Number of hidden layers (MLP)	2
Activation function	squareplus
Number of layers of message passing	1
Optimizer	ADAM
Learning rate	$1.0e^{-3}$
Batch size	100

•HGNN

Parameter	Value
Node embedding dimension	5
Edge embedding dimension	5
Hidden layer neurons (MLP)	5
Number of hidden layers (MLP)	2
Activation function	squareplus
Number of layers of message passing(pendulum)	2
Number of layers of message passing(spring)	1
Optimizer	ADAM
Learning rate	$1.0e^{-3}$
Batch size	100