Learning Rigid Body Dynamics with Lagrangian Graph Neural Network

Anonymous Author(s)
Affiliation
Address
email

Abstract

Lagrangian and Hamiltonian neural networks (LNN and HNN respectively) encode strong inductive biases that allow them to outperform other models of physical systems significantly. However, these models have, thus far, mostly been limited to simple systems such as pendulums and springs or a single rigid body such as a gyroscope or a rigid rotor. Here, we present a Lagrangian graph neural network (LGNN) that can learn the dynamics of rigid bodies by exploiting their topology. We demonstrate the performance of LGNN by learning the dynamics of ropes, chains, and trusses with the bars modeled as rigid bodies. LGNN also exhibits generalizability—LGNN trained on chains with a few segments exhibits generalizability to simulate a chain with large number of links and arbitrary link length. We also show that the LGNN can simulate unseen hybrid systems including bars and chains, on which they have not been trained on. Specifically, we show that the LGNN can be used to model the dynamics of complex real-world structures such as the stability of tensegrity structures. Finally, we discuss the non-diagonal nature of the mass matrix and it's ability to generalize in complex systems.

1 Introduction and Related Works

Movements of a robotic arm, rolling ball, or falling chain can be characterized by rigid body motion [1, 2]. Understanding the dynamics of the motion is crucial in several applications including robotics, human-robot interaction, planning, and computer graphics [3, 1]. Traditionally, the rigid body mechanics is studied in the framework of classical mechanics, which relies on either force-based or energy-based approaches [4]. Force-based approaches involve the computation of all the unknown forces based on the equations of equilibrium and hence is cumbersome for large structures. Energy-based approaches present an elegant formalism which involve the computation of a scalar quantity representing the state of a system, namely, Lagrangian ($\mathcal{L} = \mathcal{T} - \mathcal{V}$), which is the difference between the kinetic ($\mathcal{T}(q, \dot{q})$) and potential ($\mathcal{V}(q)$) energies, or Hamiltonian ($\mathcal{H} = \mathcal{T} + \mathcal{V}$), which represents the total energy of the system. This scalar quantity can, in turn, be used to predict the dynamics of the system. However, the functional form governing this scalar quantity may not be known *a priori* in many cases [5]. Thus, learning the dynamics of rigid bodies directly from the trajectory can simplify and accelerate the modeling of these systems [5, 6, 7, 8].

Learning the dynamics of particles has received much attention recently using physics-informed approaches [9]. Among these, Lagrangian neural networks (LNNs) and Hamiltonian neural networks (HNNs) are two physics-informed neural networks with strong inductive biases that outperform other learning paradigms of dynamical systems [10, 11, 12, 8, 6, 13, 7]. In this approach, a neural network is trained to learn the $\mathcal{L}$ (or $\mathcal{H}$) of a system based on its configuration $(q, \dot{q})$. The $\mathcal{L}$ is then used along with the EL equation to obtain the time evolution of the system. Note that the training of LNNs is performed by minimizing the error on the predicted trajectory with respect to the actual trajectory.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

Thus, LNNs can effectively learn the Lagrangian directly from the trajectory of a multi-particle system [6, 13].

Most of the works on LNN has focused on relatively simpler particle-based systems such as springs and pendulums [6, 13, 7, 10, 14]. This approach models a rigid body, for instance a ball, as a particle and predicts the dynamics. This approach thus ignores the additional rotational degrees of freedom of the body due to its finite volume. Specifically, while a particle in 3D has three degrees of freedom (translational), a rigid body in 3D has six degrees of freedom (translational and rotational). Thus, the dynamics and energetics associated with these degrees of motions are lost by modeling a rigid body as a particle. To the best of authors' knowledge, thus far, only one work has attempted to learn rigid body dynamics using LNNs and HNNs, where it was demonstrated the dynamics of simple rigid bodies such as a gyroscope or rotating rotor can be learned [13]. However, the LNNs used in this work, owing to their fully connected MLP architecture, are transductive in nature. An LNN trained on a double-pendulum system or 3-spring system can be used only for the same system and does not generalize to a different system size such as 3-pendulum or 5-spring, respectively. In realistic situations the number of particles in a system can vary arbitrarily, and accordingly, a large number of trained models might be required to model these systems.

An alternate approach to model these systems would be to use a graph neural network (GNN) [15, 16, 5], which, once trained, can generalize to arbitrary system sizes. GNNs have been widely used to model physical and atomic systems extensively due to their inductive bias [17, 18, 19]. GNNs have also been used to model rigid bodies mainly following two approaches, namely, *particle-based* [16] and *lumped mass* [19, 20] methods. In the first approach, a rigid body is discretized into finite number of particles and the motion of the individual particles are learned to predict the dynamics of rigid body [16]. Note that this approach is philosophically similar to mess-less methods such as smoothed-particle hydrodynamics (SPH) [21] or peridynamics (PD) [22], where the time-evolution of a continuum body is simulated by discretizing the domain using particles. This approach [16], although useful, have several limitations, namely, it does not (i) conserve physical quantities such as energy when simulated over a long duration, and (ii) generalize to a different timestep of forward simulation than the one on which it is trained. In the second approach, a rigid body is modeled as a lumped mass [19, 23], the dynamics of which is learned by assuming this lumped mass as a particle. For instance, the dynamics of a chain is modeled by discretizing the chain to smaller segments and modeling each segment as a lumped mass. As mentioned earlier, this approach leads to the loss of additional degrees of freedom that are associated with a rigid body.

Here, we present a Lagrangian graph neural network (LGNN) framework that can learn the dynamics of rigid bodies. Specifically, exploiting the topology of a physical system, we show that a rigid body can be modeled as a graph. Further, the Lagrangian of the graph structure can be learned directly by minimizing the loss on the predicted trajectory with respect to the actual trajectory of the system. The major contributions of the work are as follows.

- **Topology aware modeling of rigid body.** We present a graph-based model for rigid bodies such as in-extensible ropes, chains, or trusses. Further, we demonstrate using LGNN that the dynamics of these systems can be learned in the Lagrangian framework.
- **Generalizability to arbitrary system sizes.** We show that LGNN can generalize to arbitrary system sizes once trained.
- **Generalizability to complex unseen topology.** We demonstrate that the LGNN can generalize to unseen topology, that is, links with varying lengths, a combination of truss and chain structures, and different boundary conditions.

Altogether, we demonstrate that LGNN can be a strong framework for simulating the dynamics of rigid bodies.

2 Dynamics of Rigid Bodies

The dynamics of a physical system can be represented as $\ddot{q} = F(q, \dot{q}, t)$, where $q, \dot{q} \in \mathbb{R}^D$ is a function of time (t) for a system with D degrees of freedom. The future states or $trajectory$ of the system can be predicted by integrating these equations to obtain $q(t+1)$ and so on. While there are several physics-based methods for generating the $dynamics$ of the system such as d'Alembert's principle, Newtonian, Lagrangian, or Hamiltonian approaches, all these approaches result in the equivalent sets of equations [3].

The two broad paradigms for modeling the dynamics involve force- and energy-based approaches. Energy-based approaches is an elegant framework, which relies on the computation of a single scalar quantity, for instance energy, that represents the state of system. The dynamics of the system is, in turn, computed based on this scalar quantity. Among the energy-based approaches, Lagrangian formulation has been widely used to predict the dynamics of particles and rigid bodies by computing the Lagrangian $\mathcal{L}$ of the system. The standard form of Lagrange's equation for a system with $holonomic$ constraints is given by $\frac{d}{dt}\left(\nabla_{\dot{q}}\mathcal{L}\right) - \left(\nabla_{q}\mathcal{L}\right) = 0$, and the Lagrangian is $\mathcal{L}(q, \dot{q}, t) = \mathcal{T}(q, \dot{q}, t) - \mathcal{V}(q, t)$ with $\mathcal{T}(q, \dot{q}, t)$ and $\mathcal{V}(q, t)$ representing the total kinetic energy of the system and the potential function from which generalized forces can be derived. Accordingly, the dynamics of the system can be represented using *Euler-Lagrange (EL)* equations as $\ddot{q}_i = \left(\nabla_{\dot{q}_i \dot{q}_i}\mathcal{L}\right)^{-1}\left[\nabla_{q_i}\mathcal{L} - \left(\nabla_{\dot{q}_i q_i}\mathcal{L}\right)\dot{q}_i\right]$.

Modified Euler-Lagrange Equation. A modified version of the EL can be used in cases where some of the terms involved in the equation can be decoupled. This formulation allows explicit incorporation of constraints (holonomic and Pfaffian) and additional dissipative terms for friction or drag [3, 1]. In rigid body motion, Pfaffian constraints can be crucial in applications such as multi-fingered grasping where, the velocity of two or more fingers are constrained so that the combined geometry formed is able to catch or hold an object. A generic expression of constraints for these systems that accounts for both holonomic and Pfaffian can be $A(q)\dot{q} = 0$, where, $A(q) \in \mathbb{R}^{k \times D}$ represents k velocity constraints. In addition, drag, friction or other dissipative terms of a system can be expressed as an additional forcing term in the EL equation. It is worth noting that EL equation, by nature, is energy conserving. Hence, the additional dissipative terms are crucial for modeling realistic systems with friction and drag. If these terms are not included, the system will essentially try to simulate an energy preserving trajectory, thereby resulting in huge errors in the dynamics [14].

Considering the additional forces mentioned above, the modified EL equation can be written as:

$$\frac{d}{dt}\nabla_{\dot{q}}\mathcal{L} - \nabla_{q}\mathcal{L} + A^T(q)\lambda - \Upsilon - F = 0 \tag{1}$$

where A^T forms a non-normalized basis for the constraint forces, $\lambda \in \mathbb{R}^k$, known as the Lagrange multipliers, gives the relative magnitudes of these force constraints, Υ represents the non-conservative forces, such as friction or drag, which are not directly derivable from a potential, and F represents any external forces acting on the system. This equation can be modified to obtain $\ddot{q}$ as:

$$\ddot{q} = M^{-1}\left(-C\dot{q} + \Pi + \Upsilon - A^T(q)\lambda + F\right) \tag{2}$$

where $M = \frac{\partial}{\partial \dot{q}}\frac{\partial \mathcal{L}}{\partial \dot{q}}$ represents the mass matrix, $C = \frac{\partial}{\partial q}\frac{\partial \mathcal{L}}{\partial \dot{q}}$ represents Coriolis-like forces, and $\Pi = \frac{\partial \mathcal{L}}{\partial q}$ represents the conservative forces derivable from a potential. Differentiating the constraint equation gives $A(q)\ddot{q} + \dot{A}(q)\dot{q} = 0$. Solving for λ and substituting in Equation 2, we obtain $\ddot{q}$ as

$$\ddot{q} = M^{-1}\left(\Pi - C\dot{q} + \Upsilon - A^T(AM^{-1}A^T)^{-1}\left(AM^{-1}(\Pi - C\dot{q} + \Upsilon + F) + \dot{A}\dot{q}\right) + F\right) \tag{3}$$

For a system subjected to these forces, the dynamics can be learned using LNN by minimizing the loss on the predicted and observed trajectory, where the predicted acceleration $\hat{\ddot{q}}$ is obtained using the Equation 3. It is worth noting that in this equation, M, C, and Π can be directly derived from the $\mathcal{L}$. Constraints on the systems are generally known as they generally form part of the topology. It is worth noting that there are some recent works that focus on learning constraints as well [8].

3 Lagrangian Mechanics for Rigid Bodies

In the case of particle systems such as spring or pendulum systems, the approach mentioned in Sec.2 can be directly used in conjunction with an LNN to learn the dynamics. In this case, the mass matrix $M(q)$ remains constant with only diagonal entries m_{ii} in Cartesian coordinates. Inducing this as a prior knowledge, wherein the masses are parameterized as a diagonal matrix is shown to simplify the learning process [13]. However, in the case of a rigid body, the mass matrix is non-diagonal in the Cartesian coordinates. Further, the kinetic energy term $\mathcal{T}$ becomes a function of both position and velocity. In other words, the kinetic energy also becomes a function of the topology. This makes learning the dynamics a complex problem especially in real-world complex structures such as trusses or tensegrities, which are a combination of bars, ropes, and chains.

To this extent, we briefly review the mechanics of a falling rope or chain as an example. Note that simple rigid bodies such as a gyroscope or rotating rotor has already been studied using LNNs [13]. Of our special interest are rigid bodies that can be arbitrarily large such as chains, ropes or trusses, that can be divided into smaller constituent members. This is because, it is generally assumed that extending LNNs to large structures has been posed a challenging problem [14]. Traditionally, the mechanics of chains or ropes are modeled using discrete models [2]. Figure 1 shows a discrete model of a rope of mass M and length L. The rope is discretized into n cylindrical rods or segments each having a mass $m_i = M/n$ and length $l_i = L/n$. These segments are considered to be rigid, and with a finite uniform cross-sectional area and volume. In order to replicate realistic dynamics of a rope, the l_i should be significantly smaller that L. Note that in the case of a chain or truss, such artificial discretization is not required and the bars associated with each segment can be directly considered as a rigid body.

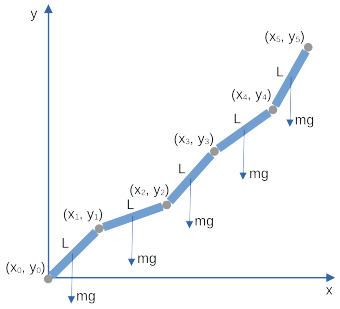

Figure 1: Chain considering rigid body dynamics. Spheres represent the connections and bars represent the links of the chain.

To formulate the $\mathcal{L}$, the generalized coordinates orientation of each link represented by $\phi_i = \frac{(y_i - y_{i-1})}{x_i - x_{i-1}}$ can be considered. Placing the origin at the beginning of first segment (see Figure 1), the center of mass of ith segment (x_i^{cm}, y_i^{cm}) can be written in terms of generalized coordinates as

$$x_i^{cm} = \sum_{j=1}^{i-1} l_j \cos\phi_j + \frac{1}{2} l_i \cos\phi_i, \qquad y_i^{cm} = \sum_{j=1}^{i-1} l_j \sin\phi_j + \frac{1}{2} l_i \sin\phi_i \tag{4}$$

Accordingly, the kinetic energy of the system is given by [2]

$$\mathcal{T} = \frac{1}{2} \sum_{i=1}^{n} \left(m_i (x_i^{cm2} + y_i^{cm2}) + I_i \dot{\phi}_i^2 \right) \tag{5}$$

where $I_i = \frac{1}{12} m_i l_i^2$ represents the moment of inertia of the rigid segment i. Similarly, the potential energy of the system can be expressed as:

$$\mathcal{V} = \sum_{i=1}^{n} m_i g y_i^{cm} \tag{6}$$

where g represents the acceleration due to gravity. Finally, the Lagrangian of the system can be obtained as $\mathcal{L} = \mathcal{T} - \mathcal{V}$, which can be substituted in the EL equation to obtain the dynamics of the rigid body.

4 Lagrangian Graph Neural Network

To learn the dynamics of rigid body, we employ the approach shown in Figure 2. Specifically, we model a physical system as a graph. Further, the Lagrangian of system is learned by decoupling the potential and kinetic energy, each of which are learned by two GNNs, namely, $\mathcal{G}_\mathcal{V}$ and $\mathcal{G}_\mathcal{T}$. Finally, the Lagrangian is computed as $\mathcal{L} = \mathcal{T} - \mathcal{V}$. This framework is trained end-to-end based by minimizing the loss on the acceleration predicted by the LGNN using EL equation with respect to the ground truth. In this section, we describe the LGNN architecture for rigid bodies in detail (See Figure 3 for an overview). We empirically show that the dynamics of a rigid body can be learned by LGNN. In addition, due to the inductive nature of the graph architecture, once trained on a small system, LGNN can generalize to arbitrary system sizes and topology.

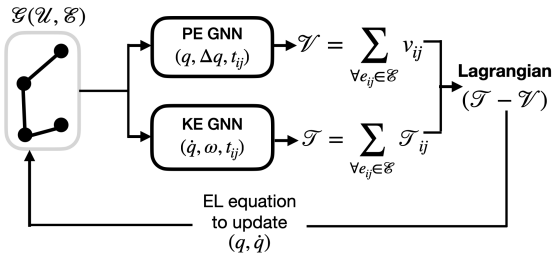

Figure 2: Learning rigid body dynamics with LGNN.

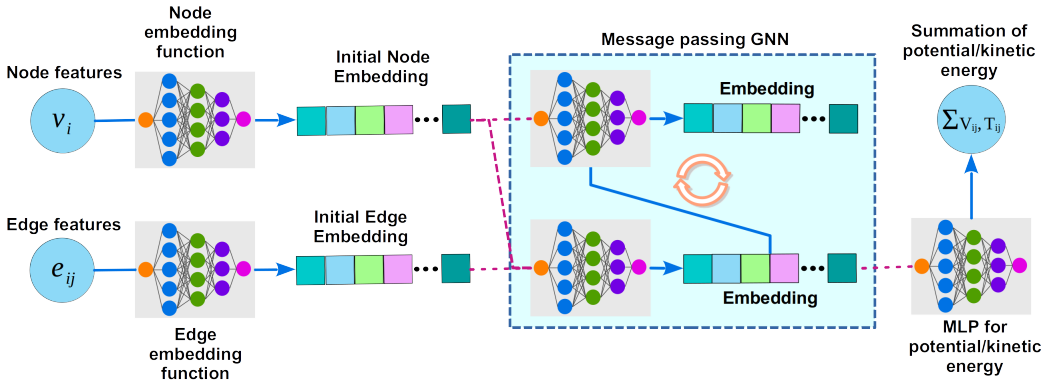

Figure 3: Architecture of LGNN for rigid bodies.

4.1 Graph architecture

Graph structure. Figure 1 shows a chain. The (undirected) graph of the physical system is constructed by considering the bars/segments of the chain as the edges and the connections as nodes. Here, edges represent the rigid bodies and node represent the connection between these rigid bodies. This is in contrast to earlier approaches used for particle-based systems, where node represented the particle position and edge represented the connections between them. Hereon, we use the notation $\mathcal{G}(\mathcal{U}, \mathcal{E})$ to to represent the graph representation of a rigid body with $\mathcal{U}$ and $\mathcal{E}$ as its node and edge sets.

Overview of the architecture. As shown in Figure 3, we use two GNNs; one to predict the potential energies and the other to predict kinetic energies. From these predictions the Lagrangian is computed. The error on the Lagrangian is minimized through an RMSE loss function to jointly train both the GNNs. The architecture of both the GNNs, shown in Figure 3, are identical.

Input features. Each node $u_i \in \mathcal{U}$ is characterized by its position $q_i = (x_i, y_i, z_i)$, and velocity $(\dot{q}_i)$. Each edge e_{ij} is characterized by its *type* t_{ij}, and the relative differences in the *positions* $(\Delta q_{ij} = q_i - q_j)$ of its connecting nodes, and $\omega_{ij} = \Delta q_{ij} \times \Delta \dot{q}_{ij}$. The type t_{ij} is a discrete variable and is useful in distinguishing edges of different characteristics within a system (Ex. moment inertia or area of cross section of the edge). Note that the velocity of a rigid body represented by an edge is a function of the velocities of its end points in two and three dimensional spaces. Hence, we do not explicitly track edge velocities.

Pre-Processing. In the pre-processing layer, we construct a dense vector representation for each node $v_i \in \mathcal{U}$ and edge $e_{ij} \in \mathcal{E}$ using MLPs (multi-layer perceptrons). The exact operation for potential energy is provided below in Eqs.7-8. For kinetic energy, we input $\dot{q}_i$ in Eq 7 instead of q_i and ω_{ij} in Eq. 8 instead of Δq_{ij}.

$$\mathbf{h}_i^0 = \texttt{squareplus}(\texttt{MLP}(q_i)) \tag{7}$$

$$\mathbf{h}_{ij}^0 = \texttt{squareplus}(\texttt{MLP}(\texttt{one-hot}(t_i), \Delta q_{ij})) \tag{8}$$

`squareplus` is an activation function.

Message passing. To infuse structural information in the edge and node embeddings, we perform L layers of message passing, wherein the embedding in each layer $l \in [1, \cdot, L]$ is computed as follows:

$$\mathbf{h}_{ij}^{l+1} = \texttt{squareplus}\left(\texttt{MLP}\left(\mathbf{h}_{ij}^l + \mathbf{W}_{\mathcal{E}}^l \cdot \left(\mathbf{h}_i^l || \mathbf{h}_j^l\right)\right)\right) \tag{9}$$

Here, $\mathbf{W}_{\mathcal{E}}^l$ is a layer-specific learnable weight vector. The node embeddings in a given layer l are learned as follows:

$$\mathbf{h}_i^{l+1} = \texttt{squareplus}\left(\texttt{MLP}\left(\mathbf{h}_i^l + \sum_{j \in \mathcal{N}_i} \mathbf{W}_{\mathcal{U}}^l \cdot \mathbf{h}_{ij}^l\right)\right) \tag{10}$$

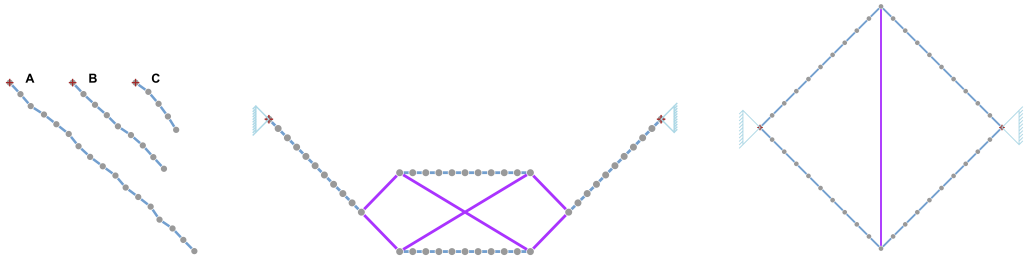

Figure 4: Topologies of the n-segment chain/rope structures with $n = 4, 8, 16$ and complex structures made with chains and rods. 4-segment system is used train the LGNN and the zero-shot generalizability is evaluated on all other systems.

Here, $\mathcal{N}_i = \{(u_i, u_j) \in \mathcal{E}\}$ denotes the edges incident on node u_i. Similar to $\mathbf{W}^l_{\mathcal{E}}$, $\mathbf{W}^l_{\mathcal{U}}$ is a layer-specific learnable weight vector, which performs a linear transformation on the embedding of each incident edge.

Following L layers of message passing, the final node and edge representations in the Lth layer are denoted as $\mathbf{z}_i = \mathbf{h}^L_i$ and $\mathbf{z}_{ij} = \mathbf{h}^L_{ij}$ respectively.

Potential and kinetic energy prediction. The predicted potential energy of each edge (rigid body) is computed by passing its final layer embedding through an MLP, i.e., $\widehat{v_{ij}} = \texttt{MLP}(\mathbf{z}_{i,j})$. The global predicted potential energy of the rigid body system is therefore the sum of the individual energies, i.e., $\widehat{\mathcal{V}} = \sum_{\forall e_{ij} \in \mathcal{E}} \widehat{v}_{ij}$. For kinetic energy, the computation is identical except that it occurs in the other GNN with parameters optimized for kinetic energy.

Loss function: The predicted Lagrangian is simply the difference between the predicted kinetic energy and the potential energy. Using Euler-Lagrange equations, we obtain the predicted acceleration $\widehat{\ddot{q}}_i(t)$ for each node u_i. The ground truth acceleration is computed directly from the ground truth trajectory using the Verlet algorithm as:

$$\ddot{q}_i(t) = \frac{1}{(\Delta t)^2}[q_i(t + \Delta t) + q_i(t - \Delta t) - 2q_i(t)] \tag{11}$$

The parameters of the GNNs are trained to minimize the RMSE loss over the entire trajectory $\mathbb{T}$:

$$\mathcal{L} = \frac{1}{|\mathcal{U}|}\left(\sum_{\forall u_i \in \mathcal{U}} \sum_{t=2}^{|\mathbb{T}|} \left(\ddot{q}_i(t) - \left(\widehat{\ddot{q}}_i(t)\right)\right)^2\right) \tag{12}$$

Since the integration of the equations of motion for the predicted trajectory is also performed using the same algorithm as: $q(t+\Delta t) = 2q(t) - q(t-\Delta t) + \ddot{q}(\Delta t)^2$, this method is equivalent to training from trajectory/positions.

4.2 Extension to Hamiltonian graph neural network

It is worth noting that the LGNN presented here can be extended directly to HGNN. Specifically, due to the modular nature of the LGNN, which predicts the kinetic and potential energies using two separate graphs, the final computation of the graph output could be modified to compute the Hamiltonian instead of the Lagrangian as $\mathcal{H} = \mathcal{T} + \mathcal{V}$.

5 Empirical Evaluation

In this section, we evaluate the ability of LGNN to learn rigid body dynamics. In addition, we evaluate the ability of LGNN to generalize to larger unseen system sizes, complex topology, and realistic structures such as tensegrity.

5.1 Experimental setup

• **Simulation environment:** All the training and forward simulations are carried out in the JAX environment [18]. The graph architecture is implemented using the jraph package [24]. All the codes

related to dataset generation and training are made available in the Supplementary.
Software packages: numpy-1.20.3, jax-0.2.24, jax-md-0.1.20, jaxlib-0.1.73, jraph-0.0.1.dev0
Hardware: Memory: 16GiB System memory, Processor: Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz

•**Baselines:** As outlined earlier, there are very few works on rigid body simulations using graph-based approaches, where the graph is used to model the topology of the rigid body. As such, to compare the performance of LGNN we employ a graph network simulator GNS, which employs a full graph network architecture [5, 12, 16] to predict the update in the position and velocity of node based on the present position and velocity. GNS has been shown to be a versatile model with the capability to simulate a wide range of physical systems [16].

• **Datasets and systems:** To evaluate the performance LGNN, we selected n-chain/rope systems, where $n = (4, 8, 16)$. All the graph based models are trained only on 4-segment chain system, which are then evaluated on other system sizes. Further, to evaluate the zero-shot generalizability of to large-scale unseen systems, we simulate 8-, and 16-segment chain systems. Further, to push the limits of LGNN, we evaluate the model trained on 4-segment chain to complex shaped topologies involving truss members (long rigid members) and chains (short rigid members), which have more than 40 segments (see Figure 4). The mass m_i and moment of inertia I_i of all the members are maintained to be the same for all the segments irrespective of their length. The details of the experimental systems are given in Appendix.

• **Evaluation Metric:** Following the work of [13], we evaluate performance by computing the relative error in **(1)** the trajectory, known as the *rollout error*, given by $RE(t) = ||\hat{q}(t) - q(t)||_2/(||\hat{q}(t)||_2 + ||q(t)||_2)$ and **(2)** *energy violation error* given by $||\hat{\mathcal{H}} - \mathcal{H}||_2/(||\hat{\mathcal{H}}||_2 + ||\mathcal{H}||_2)$. In addition, we also compute the geometric mean of rollout and energy error to compare the performance of different models [13]. 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 hat, that is x, represent the ground truth.

• **Model architecture and training setup:** For the graph architectures, namely, LGNN and GNS, all the neural networks are modeled as one hidden layer MLPs with varying number of hidden units. For all the MLPs, a square-plus activation function is used due to its double differentiability. In contrast to the earlier approaches, here, the training is not performed on trajectories. Rather, it is performed on 10000 data points generated from 100 trajectories for all the models. This dataset is divided randomly in 75:25 ratio as training and validation set. The model performance is evaluated on a forward trajectory, a task it was not explicitly trained for, of $1s$. Note that this trajectory is $\sim$2-3 orders of magnitude larger than the training trajectories from which the training data has been sampled. The dynamics of n-body system is known to be chaotic for $n \geq 2$. Hence, all the results are averaged over trajectories generated from 100 different initial conditions. Detailed model architecture associated with each of the models and the hyperparameters used in the training are provided in the Appendix.

5.2 Comparison with baselines

Model performance. Figure 5 shows the performance of LGNN in comparison to GNS. First, we analyze the performance of LGNN on the 4-link system on which it was trained. We observe that LGNN exhibits superior performance in terms of both energy conservation error and rollout error for the 4-link system in comparison to GNS. Specifically, LGNN exhibits $\sim$3 orders of magnitude better performance than GNS in terms of both energy violation error and rollout error.

Generalizability to different system sizes. Now, we analyze the performance of LGNN, trained on 4-link segment, on 8- and 16-link segments. We observe that LGNN exhibits comparable performances with respect to the 4-segment model, in terms of both energy violation error and rollout error, on systems with 8-, and 16-segments that are unseen by the model. In contrast, GNS exhibits relatively increased error in energy violation error and rollout error, with the superiority in performance of LGNN maintained by orders of magnitude. This confirms that LGNN can generalize to larger unseen system sizes when trained on a significantly smaller system size.

5.3 Zero-shot generalizability

In order to analyze the zero-shot generalizability of the trained LGNN to simulate complex real-world geometries and structures, we evaluate the ability of LGNN to model the dynamics of tensegrity structures. Note that tensegrity structures are truss-like structures comprising of both tension and

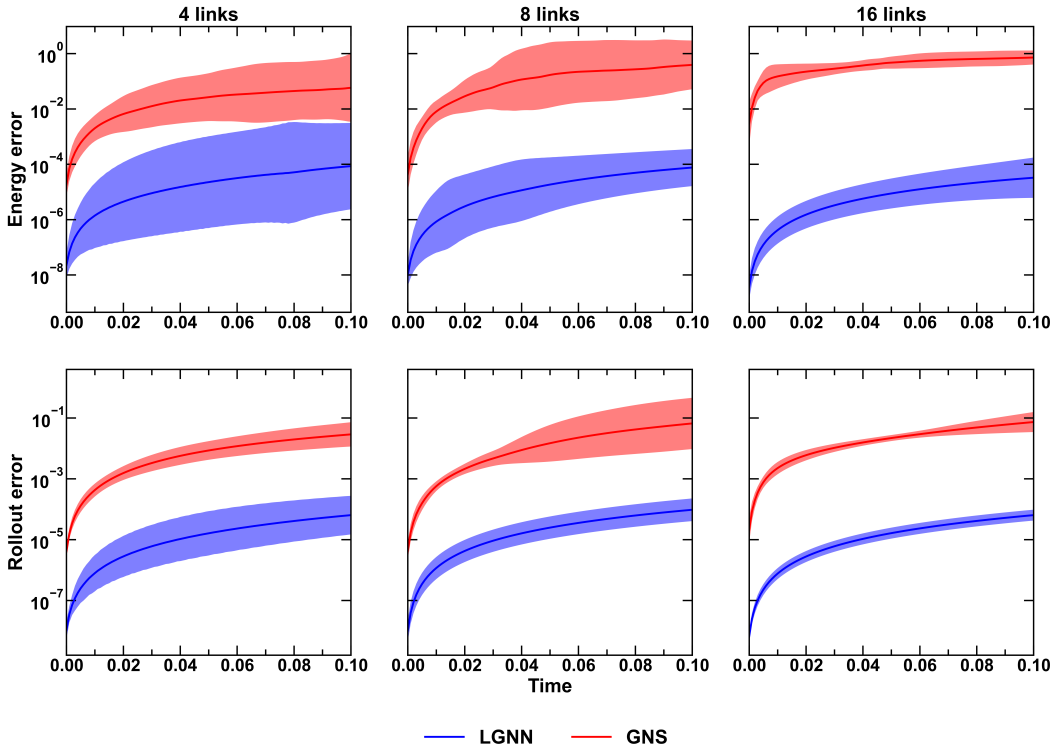

Figure 5: Energy and rollout error of $n-$link chains predicted using LGNN in comparison with ground truth for $n = 4, 8, 16$. Note that LGNN is trained only on the 4-link chain and predicted on all the systems. Shaded region shows the 95% confidence interval based on 100 forward simulations with random initial conditions.

compression-members. The topology of a tensegrity structure is designed so that the compression members are always bars and the tension members are always ropes. Here, we analyse the ability LGNN to model the equilibrium dynamics of two complex tensegrity structures shown in Figure 4.

To this extent, we use the LGNN trained on the 4-segment structure. We convert the rigid body structure to an equivalent graph and use the trained LGNN to predict the dynamics of the structure when released from the original configuration under gravity. Figure 6 shows the energy error and rollout for both the complex structures shown in Figure 4. We note that the LGNN is able to generalize to a complex structure with varying bar lengths and topology with high accuracy. Specifically, the energy violation and rollout error exhibits very low values for LGNN ($\sim 10^{-4}$). Further, it saturates after a few initial timestep suggesting an equilibrium dynamics. In contrast, we observe that the error in GNS is very high and continues to increase until it reaches 1, which is the maximum it can take. This confirms the superior nature of LGNN to generalize to arbitrary topology, boundary conditions, and bar lengths, after training on a simple 4-segment chain with constant length segments. Note that since the initial configuration for the forward simulation is fixed, it is not possible to generate error bars for the trajectory.

5.4 Nature of the learned mass matrix

Finally, we investigate the nature of the mass matrix of LGNN for different systems. Note that in earlier approaches either the mass matrix was learned directly for a given system based on the EL equations [6], or it was assumed to be diagonal in the Cartesian coordinates [13], or the functional form of kinetic energy was assumed [7]. In the present approach, we do not make any assumptions on the nature of the mass matrix. In fact, for a rigid body, the mass matrix need not be diagonal in nature and depends on the actual topology of the structure. This raises an interesting question about the nature of the mass matrix learned by the LGNN and how it generalizes to arbitrary topologies.

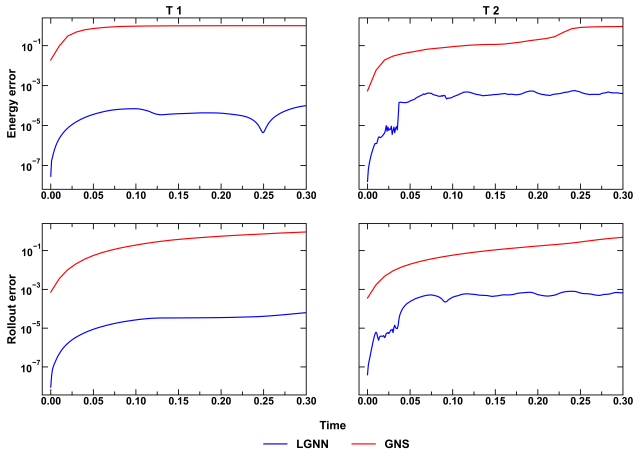

Figure 6: Energy and rollout error of systems predicted using LGNN in comparison with ground truth for T1 and T2. Note that LGNN is trained only on the 4-segment chain and predicted on all the systems. Note that since the starting configuration of the simulation is fixed, there are no error bars generated for this system.

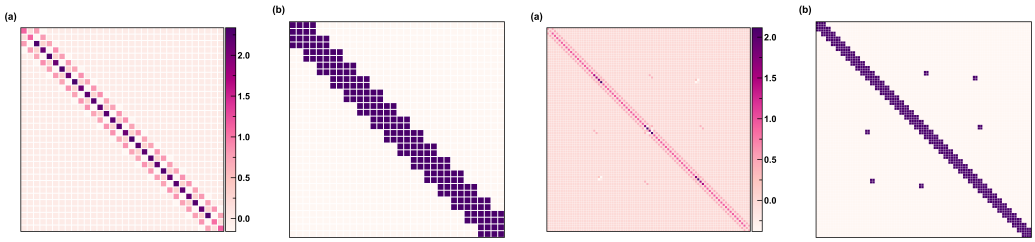

Figure 7: (a) Mass matrix and (b) binary mass matrix showing non-zero values for a chain of 16 nodes. (c) Mass matrix and (d) binary mass matrix showing non-zero values for the complex structure T1 made of chains and rods.

In order to investigate the nature of the mass matrix, we plot the mass matrix of the LGNN in Figure 7. Note that the mass matrix is computed directly from the Lagrangian as $M = \nabla_{\dot{q}\dot{q}}\mathcal{L}$, where $\mathcal{L}$ is obtained from the LGNN. First, we analyze the mass matrix of the 16-segment structure. We observe that the mass matrix is banded with a penta-diagonal band as expected for a chain structure. Now, we analyze the mass matrix for a complex structure T1. Interestingly, we observe that the mass matrix learned is non-diagonal in nature and is congruent with the complex topology of the structure (see Figure 7). This confirms that the mass matrix of LGNN is learned on-the-fly during the forward simulation that provides the versatility for LGNN to simulation complex structures.

6 Conclusions

Here, we present a LGNN-based framework that can be used to simulate the dynamics of rigid bodies. Specifically, we present the graph architecture, which allows the decoupling of kinetic and potential energies, that can be used to compute the Lagrangian of the system. We show that LGNN can learn the dynamics from a small 4-segment chain and then generalize to larger system sizes. We also demonstrate the zero-shot generalizability of LGNN to arbitrary topology including a tensegrity structure.

From the mechanics perspective, the LGNN assumes the knowledge of constraints. Learning constraints directly from the trajectory is useful. Similarly, extending LGNN to model contacts, collisions, and deformations allows more comprehesive learning of realistic systems. From the modeling perspective, in our message passing , all messages are provided equal important. Attention heads in message-passing neural networks have been shown to improve performance remarkably in several domains [25]. We plan to study the impact of attention in in our future works.

References

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

[2] Waldemar Tomaszewski and Piotr Pieranski. Dynamics of ropes and chains: I. the fall of the folded chain. *New Journal of Physics*, 7(1):45, 2005.

[3] Richard M Murray, Zexiang Li, and S Shankar Sastry. *A mathematical introduction to robotic manipulation*. CRC press, 2017.

[4] Herbert Goldstein. *Classical mechanics*. Pearson Education India, 2011.

[5] 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, 2020.

[6] 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*, 2020.

[7] 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.

[8] Shuqi Yang, Xingzhe He, and Bo Zhu. Learning physical constraints with neural projections. *Advances in Neural Information Processing Systems*, 33:5178–5189, 2020.

[9] 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.

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

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

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

[13] 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, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 13880–13889. Curran Associates, Inc., 2020.

[14] Nate Gruver, Marc Anton Finzi, Samuel Don Stanton, and Andrew Gordon Wilson. Deconstructing the inductive biases of hamiltonian neural networks. In *International Conference on Learning Representations*, 2021.

[15] 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.

[16] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pages 8459–8468. PMLR, 2020.

[17] Cheol Woo Park, Mordechai Kornbluth, Jonathan Vandermause, Chris Wolverton, Boris Kozinsky, and Jonathan P Mailoa. Accurate and scalable graph neural network force field and molecular dynamics with direct force architecture. *npj Computational Materials*, 7(1):1–9, 2021.

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

[19] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[20] Yulia Rubanova, Alvaro Sanchez-Gonzalez, Tobias Pfaff, and Peter Battaglia. Constraint-based graph network simulator. *arXiv preprint arXiv:2112.09161*, 2021.

[21] Joe J Monaghan. Smoothed particle hydrodynamics. *Annual review of astronomy and astrophysics*, 30(1):543–574, 1992.

[22] Stewart A Silling. Reformulation of elasticity theory for discontinuities and long-range forces. *Journal of the Mechanics and Physics of Solids*, 48(1):175–209, 2000.

[23] Manuel A Roehrl, Thomas A Runkler, Veronika Brandtstetter, Michel Tokic, and Stefan Obermayer. Modeling system dynamics with physics-informed neural networks based on lagrangian mechanics. *IFAC-PapersOnLine*, 53(2):9195–9200, 2020.

[24] 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.

[25] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] § 6
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] In supplementary
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] As shaded region in plots
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] In supplementary
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]