Propagation networks for model-based control under partial observation

Citation	Li, Yunzhu et al. “Propagation networks for model-based control under partial observation.” Paper presented at the 2019 International Conference on Robotics and Automation (ICRA), Montre#al, Que#bec, May 20-24, 2019, IEEE © 2019 The Author(s)
As Published	10.1109/ICRA.2019.8793509
Publisher	IEEE
Version	Original manuscript
Citable link	https://hdl.handle.net/1721.1/126583
Terms of Use	Creative Commons Attribution-Noncommercial-Share Alike
Detailed Terms	http://creativecommons.org/licenses/by-nc-sa/4.0/

Propagation Networks for Model-Based Control Under Partial Observation

Yunzhu Li, Jiajun Wu, Jun-Yan Zhu, Joshua B. Tenenbaum, Antonio Torralba, and Russ Tedrake

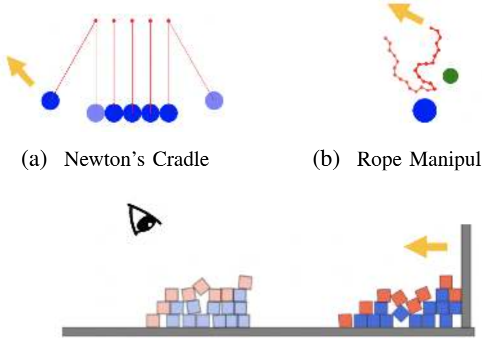
Abstract— **There has been an increasing interest in learning dynamics simulators for model-based control. Compared with off-the-shelf physics engines, a learnable simulator can quickly adapt to unseen objects, scenes, and tasks. However, existing models like interaction networks only work for fully observable systems; they also only consider pairwise interactions within a single time step, both restricting their use in practical systems. We introduce Propagation Networks (PropNet), a differentiable, learnable dynamics model that handles partially observable scenarios and enables instantaneous propagation of signals beyond pairwise interactions. Experiments show that our propagation networks not only outperform current learnable physics engines in forward simulation, but also achieve superior performance on various control tasks. Compared with existing model-free deep reinforcement learning algorithms, model-based control with propagation networks is more accurate, efficient, and generalizable to new, partially observable scenes and tasks.**

I. Introduction

Physics engines are critical for planning and control in robotics. To plan for a task, a robot may use a physics engine to simulate the effects of different actions on the environment and then select a sequence of actions to reach a desired goal configuration. The utility of the resulting action sequence depends on the accurate prediction of the physics engine; so a high-fidelity physics engine plays a critical role in robot planning. Most physics engines used in robotics, such as Mujoco [1], Bullet [2], and Drake [3], use approximate contact models, and recent studies [4], [5], [6] have demonstrated discrepancies between their predictions and real-world data. These mismatches prevent the above physics engines from solving contact-rich tasks.

Recently, researchers have started building general-purpose neural physics simulators, aiming to approximate complex physical interactions with neural networks [7], [8]. They have succeeded to model the dynamics of both rigid bodies and deformable objects (e.g., ropes). More recent work has used interaction networks for discrete and continuous control [9], [10], [11], [12].

Interaction networks, however, have two major limitations. First, interaction nets only consider pairwise interactions between objects, restricting its use in real-world scenarios, where simultaneous multi-body interactions often occur. Typical examples include Newton's cradle (Fig. 1a) or rope manipulation (Fig. 1b). Second, they need to observe the full states of a environment; however, many real-world control tasks involve dealing with partial observable states. Fig. 1c shows an example, where a robot wants to push a set of blocks into a target configuration; however, only the red blocks in the top layer are visible to the camera.


(a) Newton's Cradle

(b) Rope Manipulation

(c) Box Pushing

Fig. 1: **Challenges for existing differentiable physics simulators:** Modeling the dynamics of (a) Newton's cradle or (b) a rope requires instantaneous propagation of multi-object interaction. For (a), our goal is to control the leftmost ball so that rightmost ball hits the target (transparent). For (b), our goal is to control the rope to reach the target (transparent), while the blue and green circles are fixed obstacles. (c) Pushing a group of boxes to a target configuration requires dynamics modeling under partial observations. Here, the camera is looking down and only red blocks are observable.

In this paper, we introduce Propagation Networks (PropNet), a differentiable, learnable engine that simulates multi-body object interactions. PropNet handles partially observable situations by operating on a latent dynamics representation; it also enables instantaneous propagation of signals beyond pairwise interactions using multi-step effect propagation. Specifically, by representing a scene as a graph, where objects are the vertices and object interactions are the directed edges, we initialize and propagate the signals through the directed paths in the interaction graph at each time step.

Experiments demonstrate that PropNet consistently outperforms interaction networks in forward simulation. PropNet's ability to accurately handle partially observable states brings significant benefits for control. Compared with interaction nets and state-of-the-art model-free deep reinforcement learning algorithms, model-based control using propagation networks is more sample-efficient, accurate, and generalizes better to new, partially observable scenarios.

II. Related Work

A. Differentiable Physics Simulators

In recent years, researchers have been building differentiable physics simulators in various forms [1], [3], [13], [14].

Y. Li, J. Wu, J.-Y. Zhu, J. B. Tenenbaum, A. Torralba, and R. Tedrake are with the Computer Science and Artificial Intelligence Laboratory (CSAIL) at Massachusetts Institute of Technology, Cambridge, MA, USA

Our project page: `http://propnet.csail.mit.edu`

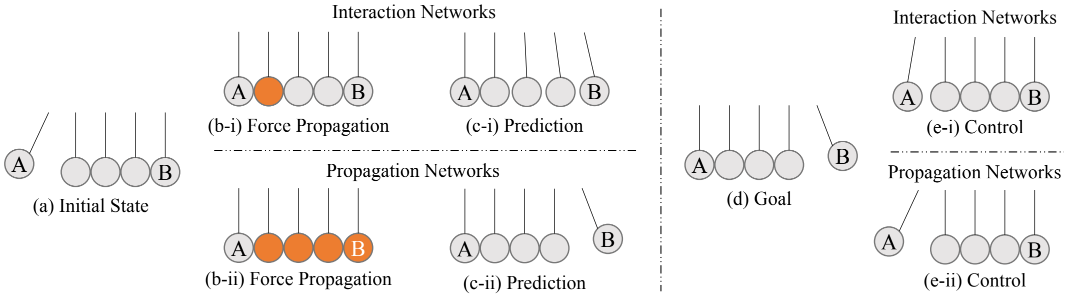


Fig. 2: **Newton's Cradle.** (a) shows the initial states of a Newton's cradle, based on which both the Interaction Networks and Propagation Networks try to predict future states; (b-i) The Interaction Networks can only propagate the force along a single relation at a time step, thus results in a false prediction (c-i); (b-ii) Our proposed method can propagate the force correctly which leads to the correct prediction (c-ii); (d) A downstream task where we aim to achieve a specific goal using the learned model; (e-i) Model-based control methods fail to produce the correct control using Interaction Networks while (e-ii) our model can provide the desired control signal.

For example, approximate, analytical differentiable rigid body simulators [14], [15] have been deployed for tool manipulation and tool-use planning [16].

Among them, two notable efforts on learning differentiable simulators include interaction networks [7] and neural physics engines [8]. These methods restrict themselves to pairwise interactions for generalizability. However, this simplification limits their ability to handle simultaneous, multi-body interactions. In this work, we tackle this problem by learning to propagate the signals multiple steps on the interaction graph. Gilmer et al. [17] have recently explored message passing networks, but with a focus on quantum chemistry.

B. Model-Predictive Control with a Learned Simulator

Recent work on model-predictive control with deep networks [18], [19], [20], [21], [22] often learns an abstract-state transition function, instead of an explicit account of environments [23], [24]. Subsequently, they use the learned model or value function to guide the training of the policy network. Instead, PropNet learns a general physics simulator that takes raw object observations (e.g., positions, velocities) as input. We then integrate it into classic trajectory optimization algorithms for control.

A few recent papers exploit the power of interaction networks for planning and control. Many of them use interaction networks to *imagine*—rolling out approximate predictions—to facilitate training a policy network [9], [10], [11]. In contrast, we use propagation networks as a learned dynamics simulator and directly optimize trajectories for continuous control. By separating model learning and control, our model generalizes better to novel scenarios. Recently, Sanchez-Gonzalez et al. [12] also explored applying interaction networks for control. Compared with them, our propagation networks can handle simultaneous multi-body interactions and deal with partially observable scenarios.

III. Learning the Dynamics

A. Preliminaries

We assume that the interactions within a physical system can be represented as a directed graph, $G = \langle O, R \rangle$, where vertices O represent the objects, and edges R correspond to their relations (Fig. 3). Graph G can be represented as

$$O = \{o_i\}_{i=1 \dots |O|} \qquad R = \{r_k\}_{k=1 \dots |R|} \tag{1}$$

Specifically, $o_i = \langle x_i, a_i^o, p_i \rangle$, where $x_i = \langle q_i, \dot{q}_i \rangle$ is the state of object i, containing its position q_i and velocity $\dot{q}_i$. a_i^o denote its attributes (e.g., mass, radius), and p_i is the external force on object i. For the relations, we have

$$r_k = \langle u_k, v_k, a_k^r \rangle, \quad 1 \leq u_k, v_k \leq |O|, \tag{2}$$

where u_k is the receiver, v_k is the sender, and a_k^r is the type and attributes of relation k (e.g., collision, spring connection).

Our goal is to build a learnable physics engine to approximate the underlying physical interactions. We can then use it to infer the system dynamics and predict the future from the observed interaction graph G:

$$G_{t+1} = \phi(G_t), \tag{3}$$

where G_t denotes the scene states at time t. We aim to learn $\phi(\cdot)$, a learnable dynamics model, to minimize $\|G_{t+1} - \phi(G_t)\|_2$.

Below we review our baseline model Interaction Networks (IN) [7]. IN is a general-purpose, learnable physics engine, performing object- and relation-centric reasoning about physics. IN defines an object function f_O and a relation function f_R to model objects and their relations in a compositional way. The future state at time $t+1$ is predicted as

$$\begin{aligned} e_{k,t} &= f_R(o_{u_k,t}, o_{v_k,t}, a_k^r), \quad k = 1 \dots |R|, \\ \hat{o}_{i,t+1} &= f_O(o_{i,t}, \sum_{k \in \mathcal{N}_i} e_{k,t}), \quad i = 1 \dots |O|, \end{aligned} \tag{4}$$

where $o_{i,t} = \langle x_{i,t}, a_i^o, p_{i,t} \rangle$ denotes object i at time t, u_k and v_k are the receiver and sender of relation r_k, and $\mathcal{N}_i$ denotes the relations where object i is the receiver.

B. Propagation Networks

IN defines a flexible and efficient model for explicit reasoning of objects and their relations in a complex system. It can handle a variable number of objects and relations and has performed well in domains like n-body systems, bouncing balls, and falling strings. However, one fundamental limitation of IN is that at every time step t, it only considers local information in the graph G and cannot handle instantaneous propagation of forces, such as Newton's cradle shown in Fig. 2, where ball A's impact produces a compression wave that propagates through the balls immediately [25]. As force propagation is a common phenomenon in rigid-body dynamics, this shortcoming has limited IN's practical applicability.

To address the above issues, we propose Propagation Networks (PropNet) to handle the instantaneous propagation of forces efficiently. Our method is inspired by message passing, a classic algorithm in graphical models [26].

1) Effect propagation: Effect propagation requires multi-step message passing along the directed edges in graph G. Forces ejected from ball A (Fig. 2) should be propagated through the connected balls to ball B within a single time step. Force propagation is hard to analyze analytically for complex scenes. Therefore, we let PropNet learn to decide whether an effect should be propagated further or withheld.

At time t, we denote the propagating effect from relation k at propagation step l as $e_{k,t}^l$, and the propagating effect from object i as $h_{i,t}^l$. Here, we have $1 \leq l \leq L$, where L is the maximum propagation steps within each step of the simulation. Propagation can be described as

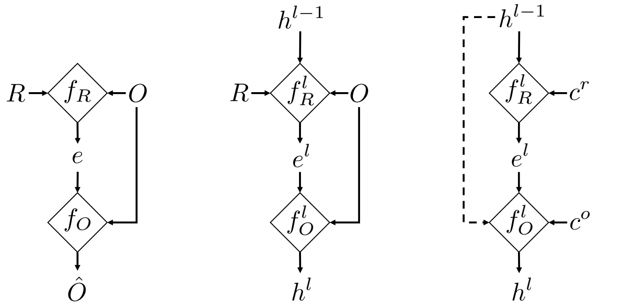
$$\begin{aligned}
&\text{Step 0:} && h_{i,t}^0 = \mathbf{0}, \quad i = 1 \ldots |O|, \\
&\text{Step } l = 1, \ldots, L: && e_{k,t}^l = f_R^l(o_{u_k,t}, o_{v_k,t}, a_k^r, h_{u_k,t}^{l-1}, h_{v_k,t}^{l-1}), \\
& && k = 1 \ldots |R|, \\
& && h_{i,t}^l = f_O^l(o_{i,t}, \sum_{k \in \mathcal{N}_i} e_{k,t}^l), \\
& && i = 1 \ldots |O|, \\
&\text{Output:} && \hat{o}_{i,t+1} = h_{i,t}^L, \quad i = 1 \ldots |O|,
\end{aligned} \tag{5}$$

where $f_O^l(\cdot)$ denotes the object propagator at propagation step l and $f_R^l(\cdot)$ denotes the relation propagator. Depending on the complexity of the task, the network weights can be shared among propagators at different propagation steps.

We name this model Vanilla PropNet. Experimental results show that the selection of L is task-specific, and usually a small L (e.g., $L = 3$) can achieve a good trade-off between the performance and efficiency.

2) Object- and relation-encoding with residual connections: We notice that Vanilla PropNet is not efficient for fast online control. As information such as states $o_{i,t}$ and attributes a_k^r are fixed at a specific time step, they can be shared without re-computation between each sequential propagation step. Hence, inspired by the ideas on fast RNNs training [27], [28], we propose to encode the shared information beforehand and reuse them along the propagation steps. We denote the encoder for objects as $f_O^{\text{enc}}(\cdot)$ and the encoder for relations as $f_R^{\text{enc}}(\cdot)$. Then,

$$c_{i,t}^o = f_O^{\text{enc}}(o_{i,t}), \qquad c_{k,t}^r = f_R^{\text{enc}}(o_{u_k,t}, o_{v_k,t}, a_k^r). \tag{6}$$



(a) IN (b) Vanilla PropNet (c) PropNet

Fig. 3: **Graphical illustration of the models.** (a) The structure of Interaction Networks as detailed in Eqn. 4; (b) The internal structure of Vanilla PropNet is described in Eqn. 5, where the effects e^l and h^l are propagated through the propagators f_O^l and f_R^l along the directed relations in the graph G; (c) The shared object encoding c^o and relation encoding c^r are inputs to the internal modules, where there are also residual connections for better effect propagation as described in Eqn. 6 and 7.

$$\begin{tikzcd}
G_t \arrow[r, "\phi"] & G_{t+1}
\end{tikzcd}$$

(a) Full-state observation

$$\begin{tikzcd}
\psi(g(G_t)) \arrow[r, dashed] & G_t \arrow[d] & \psi(g(G_{t+1})) \arrow[r, dashed] & G_{t+1} \arrow[d] \\
& g(G_t) \arrow[ul] \arrow[d, "\omega"'] & & g(G_{t+1}) \arrow[ul] \arrow[d, "\omega"'] \\
& \tau(G_t) \arrow[rr, "\phi"] & & \tau(G_{t+1})
\end{tikzcd}$$

(b) Partial observation

Fig. 4: **Comparison between fully- and partially-observable scenarios.** (a) Forward model for fully observable environments (Eqn. 3). (b) For partially observable scenarios, we first map the observation to a latent space using function $\tau(\cdot)$, and then specify the forward dynamics over the latent space using $\phi(\cdot)$ as described in Eqn. 8. $\tau(\cdot)$ consists of $g(\cdot)$ and $\omega(\cdot)$, where $g(\cdot)$ maps the observation to object-based representations, which are then aggregated to a global representation using $\omega(\cdot)$. A decoding function $\psi(\cdot)$ maps the encoding back to the original observation space to ensure a nontrivial encoding.

In practice, we add residual links [29] between adjacent propagation steps that connect $h_{i,t}^l$ and $h_{i,t}^{l-1}$. This helps address gradient vanishing and exploding problem, and provides access to historical effects. The update rules become

$$\begin{aligned}
e_{k,t}^l &= f_R^l(c_{k,t}^r, h_{u_k,t}^{l-1}, h_{v_k,t}^{l-1}), \\
h_{i,t}^l &= f_O^l(c_{i,t}^o, \sum_{k \in \mathcal{N}_i} e_{k,t}^l, h_{i,t}^{l-1}),
\end{aligned} \tag{7}$$

where propagators $f_O^l(\cdot)$ and $f_R^l(\cdot)$ now take a new sets of inputs, which is different from Vanilla PropNet.

Based on the assumption that the effects between propagation steps can be represented as simple transformations (e.g., identity-mapping in Newton's cradle), we can use small networks as function approximators for the propagators $f_O^l(\cdot)$ and $f_R^l(\cdot)$ for better efficiency. We name this updated model Propagation Networks (PropNet).

C. Partially Observable Scenarios

For many real-world situations, however, it is often hard or impossible to estimate the full state of environments. We extend Eqn. 3 using PropNets to handle such partially observable cases by operating on a latent dynamics model:

$$\tau(G_{t+1}) = \phi(\tau(G_t)), \tag{8}$$

where $\tau(\cdot)$ is an encoding function that maps the current observation to a latent representation. As shown in Figure 4b, $\tau(\cdot)$ consists of two parts: first, PropNets $g(\cdot)$ that map the current observation to object-centric representations; second, $\omega(\cdot)$ that aggregates the object-centric representations into a fixed-dimensional global representation. We use a global representation for partially observable cases, because the number and set of observable objects vary over time, making it hard to define object-centric dynamics. In fully observable environments, $\tau(\cdot)$ reduces to an identity mapping and the dynamics is defined on the object level over the state space (Eqn. 3 and Fig. 4a). To train such a latent dynamics model, we seek to minimize the loss function: $\mathcal{L}_{\text{forward}} = \|\tau(G_{t+1}) - \phi(\tau(G_t))\|_2$.

In practice, we use a small history window of length T_{history} for the state representation, i.e., the input to $\phi(\cdot)$ is the concatenation of $\tau(G_t), \tau(G_{t-1}), ..., \tau(G_{t-T_{\text{history}}+1})$.

Using the above loss alone leads to trivial solutions such as $\phi(x) = \tau(x) = 0$ for any valid x. We tackle this based on an intuitive idea: an ideal encoding function $\tau(\cdot)$ should be able to reserve information about the scene observation. Hence, we use an aggregation function $\omega(\cdot)$ that has no learnable parameters like summation or average and introduce a decoding function $\psi(\cdot)$ to ensure a nontrivial $\tau(\cdot)$ by minimizing an additional auto-encoder reconstruction loss [30]: $\mathcal{L}_{\text{encode}} = \|G - \psi(g(G))\|_2$, where $\psi(\cdot)$ is realized as PropNets. The full model is shown in Figure 4b.

IV. Control Using Learned Dynamics

Compared to model-free approaches, model-based methods offer many advantages, such as generalization and sample efficiency, as it can approximate the policy gradient or value estimation without exhausted trials and errors.

However, an accurate model of the environment is often hard to specify and brings significant computational costs for even a single-step forward simulation. It would be desirable to learn to approximate the underlying dynamics from data.

A learned dynamics model is naturally differentiable. Given the model and a desired goal, we can perform forward simulation, optimizing the control inputs by minimizing a loss between simulated results and a goal. The model can also estimate the uncertain attributes online by minimizing the difference between predicted future and actual outcome. Alg. 1 outlines our control algorithm, which provides a natural testbed for evaluating the dynamics models.

a) Model predictive control using shooting methods: Let $\mathcal{G}_g$ be our goal and $\hat{u}_{1:T}$ be the control inputs (decision variables), where T is the time horizon. These task-specific control inputs are part of the dynamics graph. Typical choices include observable objects' initial velocity/position and external forces/attributes on objects/relations. We denote the graph encoding as $G^\tau = \tau(G)$, and the resulting trajectory after applying the control inputs as $\mathcal{G} = \{G_i^\tau\}_{i=1:T}$. The task here is to determine the control inputs by minimizing the gap between the actual outcome and the specified goal $\mathcal{L}_{\text{goal}}(\mathcal{G}, \mathcal{G}_g)$.

Algorithm 1 Control on Learned Dynamics at Time Step t

```
**Input:** Learned forward dynamics model $\phi(\cdot)$
    Predicted dynamics graph encoding $\hat{G}_t^\tau$
    Current dynamics graph encoding $G_t^\tau$
    Goal $\mathcal{G}_g$, current estimation of the attributes $A$
    Current control inputs $\hat{u}_{t:T}$
    States history $\bar{\mathcal{G}} = \{G_i^\tau\}_{i=1...t}$
    Time horizon $T$
**Output:** Controls $\hat{u}_{t:T}$, predicted next time step $\hat{G}_{t+1}^\tau$

Update $A$ by descending with the gradients
    $\nabla_A \mathcal{L}_{\text{state}}(\hat{G}_t^\tau, G_t^\tau)$
Forward simulation using the current graph encoding
    $\hat{G}_{t+1}^\tau \leftarrow \phi(G_t^\tau)$
Make a buffer for storing the simulation results
    $\mathcal{G} \leftarrow \bar{\mathcal{G}} \cup \hat{G}_{t+1}^\tau$
**for** $i = t+1, ..., T-1$ **do**
    Forward simulation
        $\hat{G}_{i+1}^\tau \leftarrow \phi(\hat{G}_i^\tau)$; $\mathcal{G} \leftarrow \mathcal{G} \cup \hat{G}_{i+1}^\tau$
**end for**
Update $\hat{u}_{t:T}$ by descending with the gradients
    $\nabla_{\hat{u}_{t:T}} \mathcal{L}_{\text{goal}}(\mathcal{G}, \mathcal{G}_g)$

Return $\hat{u}_{t:T}$ and $\hat{G}_{t+1}^\tau \leftarrow \phi(G_t^\tau)$
```

Our propagation networks can do forward simulation by taking the dynamics graph at time t as input, and produce the graph at next time step, $\hat{G}_{t+1}^\tau = \phi(G_t^\tau)$. Let's denote the forward simulation from time step t as $\hat{\mathcal{G}} = \{\hat{G}_i^\tau\}_{i=t+1...T}$ and the history until time t as $\bar{\mathcal{G}} = \{G_i^\tau\}_{i=1...t}$. We can back-propagate from the loss $\mathcal{L}_g(\bar{\mathcal{G}} \cup \hat{\mathcal{G}}, \mathcal{G}_g)$ and use stochastic gradient descent (SGD) to update the control inputs. This is known as the shooting method in trajectory optimization [31].

If the time horizon T is too long, the learned model might deviate from the ground truth due to accumulated prediction errors. Hence, we use Model-Predictive Control (MPC) [32] to stabilize the trajectory by doing forward simulation at every time step as a way to compensate the simulation error.

b) Online adaptation: In many situations, inherent attributes such as masses, friction, and damping are not directly observable. Instead, we can interact with the objects and use PropNet to estimate these attributes online (denoted as A) with SGD updates by minimizing the difference between the predicted future states and the actual future states $\mathcal{L}_{\text{state}}(\hat{G}_t^\tau, G_t^\tau)$.

V. Experiments

In this section, we evaluate the performance of our model on both simulation and control in three scenarios: Newton's Cradle, Rope Manipulation, and Box Pushing. We also test how the model generalizes to new scenarios and how it learns to adapt online.

A. Physics Simulation

We aim to predict the future states of physical systems. We first describe the network used across tasks and then present the setup of each task as well as the experimental results.

a) Model architecture: For the IN baseline, we use the same network as described in the original work [7]. For Vanilla PropNet, we adopt similar network structure where the relation propagator $f_R^l(\cdot)(1 \leq l \leq L)$ is an MLP with four 150-dim hidden layers and the object propagator $f_O^l(\cdot)(1 \leq l \leq L-1)$ has one 100-dim hidden layer. Both output a 100-dim propagation vector. For fully observable scenarios, $f_O^L(\cdot)$ has one 100-dim hidden layer and outputs a 2-dim vector representing the velocity at the next time step. For partially observable cases, $f_O^L(\cdot)$ outputs one 100-dim vector as the latent representation.

For PropNet, we use an MLP with three 150-dim hidden layers as the relation encoder $f_R^{\text{enc}}(\cdot)$ and one 100-dim hidden layer MLP as the object encoder $f_O^{\text{enc}}(\cdot)$. Light-weight neural networks are used for the propagators $f_O^l(\cdot)$ and $f_R^l(\cdot)$, both of which only contain one 100-dim hidden layer.

b) Newton's cradle: A typical Newton's cradle consists of a series of identically sized rigid balls suspended from a frame. When one ball at the end is lifted and released, it strikes the stationary balls. Forces will transmit through the stationary balls and push the last ball upward immediately. In our fully observable setup, the graph G of n balls has $2n$ objects representing the balls and the corresponding fixed pinpoints above the balls, as shown in Fig. 2a, where $n = 5$. There will be $2n$ directed relations describing the rigid connections between the fixed points and the balls. Collisions between adjacent balls introduce another $2(n-1)$ relations.

We generated 2,000 rollouts over 1,000 time steps, of which 85% of the rollouts are randomly chosen as the training set, while the rest are held as the validation set. The model was trained with a mini-batch of 32 using Adam optimizer [33] with an initial learning rate of 1e-3. We reduce the learning rate by 0.8 each time the validation error stops decreasing for over 20 epochs.

Fig. 2a-c show some qualitative results, where we compare IN and PropNet. IN cannot propagate the forces properly: the rightmost ball starts to swing up before the first collision happens. Quantitative results also show that our method significantly outperforms IN in tracking object positions. For 1,000 forward steps, IN results in an MSE of 336.46, whereas PropNet achieves an MSE of 7.85.

c) Rope manipulation: We then manipulate a particle-based rope in a 2D plane using a spring-mass model, where one end of the rope is fixed to a random point near the center and the rest of the rope is free to move. Two circular obstacles are placed at random positions near the rope and are fixed to the ground. Random forces are applied to the masses on the rope and the rope is moving in compliant with the forces. More specifically, for a rope containing n particles, there will be a total of $n+2$ objects. Each pair of adjacent masses will have spring relations connecting each other, resulting in $2(n-1)$ directed edges in the dynamics graph G. Each mass will have a collision relation with each fixed obstacle, which adds to the graph another $4n$ edges. Frictional force applied to each mass is modeled as a directed edge connecting the mass itself.

We use the same network as described above and generate 5,000 rollouts over 100 time steps. Fig. 5a and Fig. 6a show qualitative and quantitative results, respectively. We train the models with a 15-dim rope and evaluated in situations where the rope length can vary between 10 and 20. As can be seen from the figures, although the length of the underlying force propagation is fewer than Newton's Cradle's, our proposed method can still track the ground truth much more accurately and outperform IN by a large margin.

d) Box pushing: In this case, we are pushing a pile of boxes forward (Fig. 5c). We place a camera at the top of the scene, and only red boxes are observable. More challengingly, the observable boxes are not tracked. Therefore, the visibility of a specific box might change over time. The vertices in the graph are then defined as the state of the observable boxes and edges are defined as directional relations connecting every pair of observable boxes. Specifically, if there are n observable boxes, $n(n-1)$ edges are automatically generated. The dynamics function $\phi(\cdot)$ then takes both the scene representation and the action (i.e., position and velocity of the pusher) as input to perform an implicit forward simulation. As it is hard to explicitly evaluate a latent dynamics model, we evaluate the downstream control tasks instead.

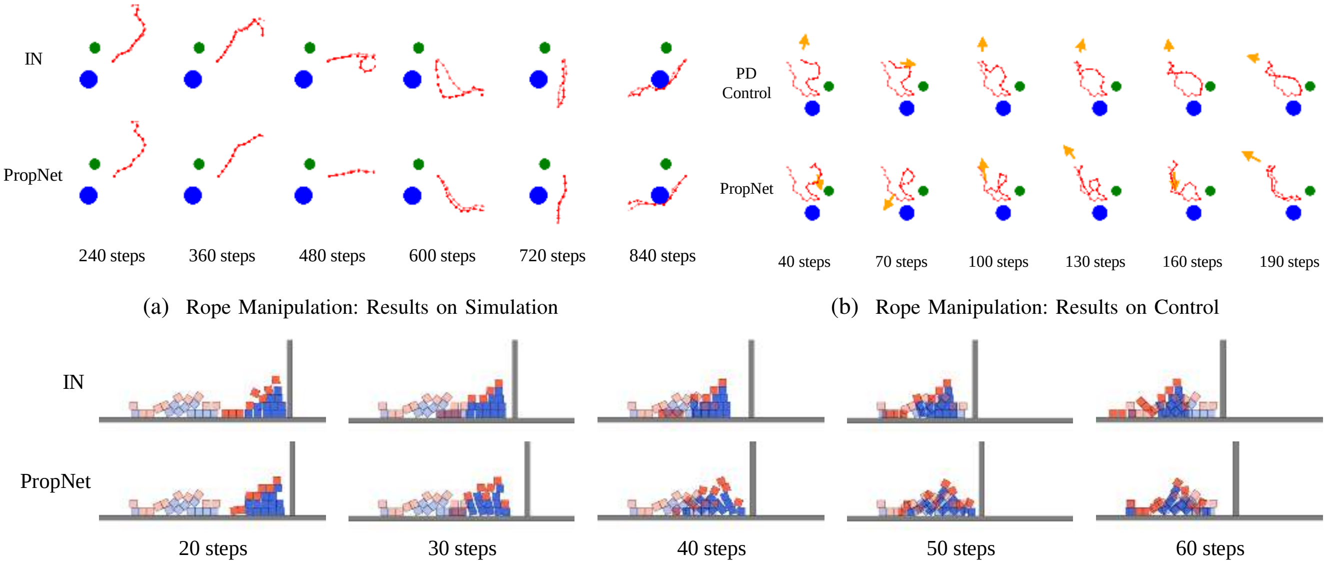
e) Ablation studies: We also provide ablation studies on how the number of propagation steps L influences the final performance. Empirically, a larger L can model a longer propagation path. They are however harder to train and more likely to overfit the training set, often leading to poor generalization. Fig. 6a and 6b show the ablation studies regarding the choice of L. PropNet achieves a high accuracy at $L = 3$, with a good trade-off between speed and accuracy. Vanilla PropNet achieves its best accuracy at $L = 2$ but generalizes less well as L increases further. This shows the benefits of using the shared encoding and residual connections used in PropNet, as described in Section III-B.2.

B. Control

We now evaluate the applicability of the learned model on control tasks. We first describe the three tasks: Newton's Cradle, Rope Manipulation, and Box Pushing, which include both open-loop and feedback continuous control tasks, as well as fully and partially observable environments. We evaluate the performance against various baselines and test its ability on generalization and online adaptation.

a) Newton's Cradle: In this scenario, we assume full-state observation and a control task would be to determine the initial angle of the left-most ball, so as to let the right-most ball achieve a specific height, which can be solved with an accurate forward simulation model.

This is an open-loop control task where we only have control over the initial condition. We thus use a simplified version of Alg. 1. Given the initial physics graph and a learned dynamics model, we iteratively do forward simulation and update the control inputs by minimizing the loss function $\mathcal{L}_{\text{goal}}(\mathcal{G}, \mathcal{G}_g)$. In this specific task, the loss $\mathcal{L}_{\text{goal}}$ is the $\mathcal{L}_2$

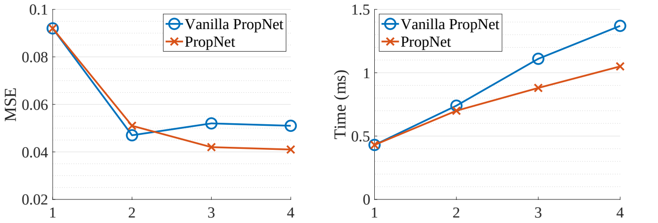


(a) Rope Manipulation: Results on Simulation

(b) Rope Manipulation: Results on Control

(c) Box Pushing: Results on Control

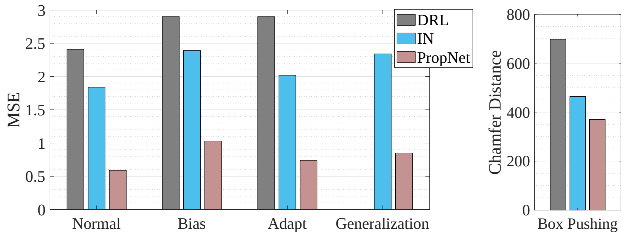
Fig. 5: **Qualitative results on simulation and control.** (a) Results on the planar rope simulation, where every mass on the rope has been applied a random force and the rope is moving in the planar in compliant with the forces. Our model better matches the ground truth and suffers less from the drifting problem as time horizon becomes longer. Here the transparent trajectories indicate the ground truth. (b) The rope manipulation task defines a continuous control problem which is to achieve a specified goal configuration by applying forces to the top two masses on the free end of the rope. The applied forces are visualized as yellow arrows and the goal configuration is shown as transparent. Note that instead of naively trying to match the top two masses (PD control), our control method based on PropNet can achieve the goal configuration by exploring the rich dynamics of the rope. (c) The box pushing task requires solving a control problem under partial observation (only red blocks are observable). The goal configuration is shown as transparent. Doing control with our propagation networks achieves more accurate outcome than with an IN. Please also see the supplementary video.



(a) Rope: Velocity

(b) Rope: Forward Time

Fig. 6: **Quantitative results on rope simulation.** We vary the propagation steps L between 2 to 4 for Vanilla PropNet and PropNet, which shows a trade-off between accuracy and efficiency. When $L = 1$, both models reduce to Interaction Networks (IN).



(a) Rope Manipulation

(b) Box Pushing

Fig. 7: **Quantitative results on control tasks.** (a) For rope manipulation, the algorithms attempt to match a specific configuration under situations where the ground-truth attributes are known ("Normal"), where the value of the attributes are unknown ("Bias"), where algorithms actively estimate these attributes online ("Adapt"), and where ropes are of varied length between 10 to 20 when the model is only trained on ropes of length 15 ("Genearlize"). DRL has the same performance for "Bias" and "Adapt" as it is model-free; it requires a fixed length input, and thus cannot generalize to ropes of a different length. (b) For box pushing, propagation networks again outperforms the other methods.

distance between the target height of the right-most ball and the highest height that has been achieved in $\mathcal{G}$.

We initialize the swing up angle as 45° and then optimize the angle with a learning rate of 0.1 for 50 iterations using Adam optimizer. We compare our model with IN. Qualitative results are shown in Fig. 2e. Quantitatively, PropNet's output angle has an MSE of 3.08 from the ground truth initial angle, while the MSE for interaction nets is 296.66.

b) Rope Manipulation: Here we define the task as to move the rope to a target configuration, where the only controls are the top two masses at the moving end of the rope (Fig. 5b). The controller tries to match the target configuration by "swinging" the rope, which requires to leverage the dynamics of the rope. The loss $\mathcal{L}_{\text{goal}}$ here is the $\mathcal{L}_2$ distance between the resulting configuration and the goal configuration.

We first assume the attributes of the physics graph is known (e.g., mass, friction, damping) and compare the performance between Proportional-Derivative controller (PD) [34], Model-free Deep Reinforcement Learning (Actor-Critic method optimized with PPO [35] - DRL), as well as Interaction Networks (IN) and Propagation Networks (PropNet) with Alg. 1. Fig. 7 shows quantitative results, where bars marked as "Normal" are the results in this task (a hand-tuned PD controller has an MSE of 2.50). PropNet outperforms the competing baselines. Fig. 5b shows a qualitative sample. Compared with the PD controller, our method leverages the dynamics and manages to match the target, instead of naively matching the free end of the rope.

We then consider situations where some of the attributes are unknown and can only be guessed before actually interacting with the objects. We randomly add noise of 15% of the original scale to the attributes as the initial guesses. The "Bias" bars in Fig. 7 show that models trained with ground-truth attributes will encounter performance drop when the supplied attributes are not accurate. However, model-based methods can do online adaptation using the actual output from the environment as feedback to correct the attribute estimation. By updating the estimated attributes over the first 20 steps of the time horizon with standard SGD, we can improve the manipulation performance so as to catch up with the situations where attributes are accurate (bars marked as "Adapt" in Fig. 7).

We further test whether our model generalizes to new scenarios, where the length of the rope is varied between 10 to 20. As can be seen in Fig. 7, our proposed method can still achieve a good performance, even though the original PropNet is only trained in situations with a fixed length 15 (PD has an MSE of 2.72 for generalization).

c) Box Pushing: In this case, we aim to push a pile of boxes to a target configuration within a predefined time horizon (Fig. 5c). We assume partial observation where a camera is placed at the top of the scene, and we can only observe the states of the boxes marked in red. The model trained with partial observation is compared with two baselines: DRL and IN. The loss function $\mathcal{L}_{\text{goal}}$ used for MPC is the $\mathcal{L}_2$ distance between the resulting scene encoding and the target scene encoding.

We evaluate the performance by the Chamfer Distance (CD) [36] between the observable boxes at the end of the episode and the target configurations, where for each box in each set, CD finds the nearest box in the other set, and sums the distance up. The negative of the distance is used as the reward for DRL. Fig. 5c and Fig. 7b show qualitative and quantitative results, respectively. Our method outperforms the baselines due to its explicit modeling of the dynamics and its ability to handle multi-object interactions.

VI. Conclusion

We have presented propagation networks (PropNet), a general learnable physics engine that outperforms the previous state-of-the-art with a large margin. We have also demonstrated PropNet's applicability in model-based control under both fully and partially observable environments. With propagation steps, PropNet can propagate the effects along relations and model the dynamics of long-range interactions within a single time step. We have also proposed to improve PropNet's efficiency by adding residual connections and shared encoding.

Acknowledgement

This work was supported by: Draper Laboratory Incorporated, Sponsor Award No. SC001-0000001002; NASA - Johnson Space Center, Sponsor Award No. NNX16AC49A; NSF #1524817; DARPA XAI program FA8750-18-C000; ONR MURI N00014-16-1-2007; and Facebook.

References

[1] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *IROS*. IEEE, 2012, pp. 5026–5033.

[2] E. Coumans, "Bullet physics engine," *Open Source Software: http://bulletphysics. org*, 2010.

[3] R. Tedrake and the Drake Development Team, "Drake: Model-based design and verification for robotics," 2019. [Online]. Available: https://drake.mit.edu

[4] R. Kolbert, N. Chavan Dafle, and A. Rodriguez, "Experimental Validation of Contact Dynamics for In-Hand Manipulation," in *ISER*, 2016.

[5] K.-T. Yu, M. Bauza, N. Fazeli, and A. Rodriguez, "More than a million ways to be pushed. a high-fidelity experimental dataset of planar pushing," in *IROS*. IEEE, 2016, pp. 30–37.

[6] N. Fazeli, S. Zapolsky, E. Drumwright, and A. Rodriguez, "Fundamental limitations in performance and interpretability of common planar rigid-body contact models," in *ISRR*, 2017.

[7] P. W. Battaglia, R. Pascanu, M. Lai, D. Rezende, and K. Kavukcuoglu, "Interaction networks for learning about objects, relations and physics," in *NeurIPS*, 2016.

[8] M. B. Chang, T. Ullman, A. Torralba, and J. B. Tenenbaum, "A compositional object-based approach to learning physical dynamics," in *ICLR*, 2017.

[9] S. Racanière, T. Weber, D. Reichert, L. Buesing, A. Guez, D. J. Rezende, A. P. Badia, O. Vinyals, N. Heess, Y. Li, R. Pascanu, P. Battaglia, D. Silver, and D. Wierstra, "Imagination-augmented agents for deep reinforcement learning," in *NeurIPS*, 2017.

[10] J. B. Hamrick, A. J. Ballard, R. Pascanu, O. Vinyals, N. Heess, and P. W. Battaglia, "Metacontrol for adaptive imagination-based optimization," in *ICLR*, 2017.

[11] R. Pascanu, Y. Li, O. Vinyals, N. Heess, L. Buesing, S. Racanière, D. Reichert, T. Weber, D. Wierstra, and P. Battaglia, "Learning model-based planning from scratch," *arXiv:1707.06170*, 2017.

[12] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia, "Graph networks as learnable physics engines for inference and control," in *ICML*, 2018.

[13] S. Ehrhardt, A. Monszpart, N. Mitra, and A. Vedaldi, "Taking visual motion prediction to new heightfields," *arXiv:1712.09448*, 2017.

[14] J. Degrave, M. Hermans, and J. Dambre, "A differentiable physics engine for deep learning in robotics," in *ICLR Workshop*, 2016.

[15] F. de Avila Belbute-Peres, K. A. Smith, K. Allen, J. B. Tenenbaum, and J. Z. Kolter, "End-to-end differentiable physics for learning and control," in *Neural Information Processing Systems*, 2018.

[16] M. Toussaint, K. Allen, K. Smith, and J. Tenenbaum, "Differentiable physics and stable modes for tool-use and manipulation planning," in *RSS*, 2018.

[17] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *ICML*, 2017.

[18] I. Lenz, R. A. Knepper, and A. Saxena, "Deepmpc: Learning deep latent features for model predictive control," in *RSS*, 2015.

[19] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep q-learning with model-based acceleration," in *ICML*, 2016.

[20] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," in *ICRA*, 2018.

[21] G. Farquhar, T. Rocktäschel, M. Igl, and S. Whiteson, "Treeqn and atreec: Differentiable tree planning for deep reinforcement learning," in *ICLR*, 2018.

[22] A. Srinivas, A. Jabri, P. Abbeel, S. Levine, and C. Finn, "Universal planning networks," in *ICML*, 2018.

[23] D. Silver, H. van Hasselt, M. Hessel, T. Schaul, A. Guez, T. Harley, G. Dulac-Arnold, D. Reichert, N. Rabinowitz, A. Barreto, and T. Degris, "The predictron: End-to-end learning and planning," in *ICML*, 2017.

[24] J. Oh, S. Singh, and H. Lee, "Value prediction network," in *NeurIPS*, 2017.

[25] D. E. Stewart, "Rigid-body dynamics with friction and impact," *SIAM review*, vol. 42, no. 1, pp. 3–39, 2000.

[26] J. Pearl, *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier, 2014.

[27] T. Lei and Y. Zhang, "Training rnns as fast as cnns," *arXiv preprint arXiv:1709.02755*, 2017.

[28] J. Bradbury, S. Merity, C. Xiong, and R. Socher, "Quasi-recurrent neural networks," in *ICLR*, 2017.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[30] G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” *Sci.*, vol. 313, no. 5786, pp. 504–507, 2006.

[31] R. Tedrake, “Underactuated robotics: Learning, planning, and control for efficient and agile machines course notes for mit 6.832,” 2009.

[32] E. F. Camacho and C. B. Alba, *Model predictive control*. Springer Science & Business Media, 2013.

[33] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *ICLR*, 2015.

[34] K. J. Åström and T. Hägglund, *PID controllers: theory, design, and tuning*. Instrument society of America Research Triangle Park, NC, 1995, vol. 2.

[35] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv:1707.06347*, 2017.

[36] H. G. Barrow, J. M. Tenenbaum, R. C. Bolles, and H. C. Wolf, “Parametric correspondence and chamfer matching: Two new techniques for image matching,” in *IJCAI*, 1977.