Propagation Networks for Model-Based Control
Under Partial Observation

Yunzhu Li¹, Jiajun Wu¹, Jun-Yan Zhu¹, Joshua B. Tenenbaum¹, Antonio Torralba¹, and Russ Tedrake¹

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. With these innovations, our propagation networks not only outperform current learnable physics engines in forward simulation, but also achieves superior performance on various control tasks. Compared with existing deep reinforcement learning algorithms, model-based control with propagation networks is more accurate, efficient, and generalizable to novel, 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 accuracy of the physics engine’s predictions; so a high-fidelity physics engine plays an important role in robot planning. Most physics engines used in robotics (such as Mujoco [1] and Bullet [1]) use approximate contact models, and recent studies [2], [3], [4] have demonstrated discrepancies between their predictions and real-world data. These mismatches make contact-rich tasks hard to solve with these physics engines.

Recently, researchers have started building general-purpose neural physics simulators, aiming to approximate complex physical interactions with neural networks [5], [6]. They have succeeded to model the dynamics of both rigid-bodies and deformable objects (e.g., strings). More recent work has used interaction networks for discrete and continuous control [7], [8], [9], [10].

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 string manipulation (Fig. 1b). Second, they need to observe the full states of the environment; however, many real-world control tasks involve dealing with partial observable states.

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 the 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 outperform 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 novel, partially observable scenarios.

Fig. 1 shows an example, where the robot wants to push a set of blocks into a target configuration; however, only the red blocks on the surface are visible to the camera.

II. RELATED WORK

A. Differentiable Physics Simulators

In recent years, researchers have been building differentiable physics simulators in various forms [11], [12]. For example, approximate, analytical differentiable rigid body
simulators [12], [13] have been deployed for tool manipulation and tool-use planning [14].

Among them, two notable efforts on learning differentiable simulators include interaction networks [5] and neural physics engines [6]. These methods restrict themselves to pairwise interactions for generalizability. However, this limits their ability to handle simultaneous, multi-body interactions. In this paper, we tackle this problem by learning to propagate the signals according to the interaction graph. Gilmer et al. [15] 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 [16], [17], [18], [19], [20] often learns an abstract-state transition function, instead of an explicit account of the environment [21], [22]. 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.

There have been a few papers that 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 [7], [8], [9]. 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. [10] 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 relations (Fig. 3). Graph $G$ can be represented as

$$O = \{ o_i \}_{i=1, \ldots, |O|} \quad R = \{ r_k \}_{k=1, \ldots, |R|} \quad (1)$$

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

$$r_k = \langle u_k, v_k, a_k \rangle, \quad 1 \leq u_k, v_k \leq |O|, \quad (2)$$

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

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

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

where $G_t$ denotes the scene states at time $t$ and $\phi$ is a learnable dynamics model.

Below we review our baseline model Interaction Networks (IN) [5]. 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

$$e_{k,t} = f_R(o_{uk,t}, o_{vk,t}, a_k^t), \quad k = 1, \ldots, |R|, \quad (4)$$

$$\hat{e}_{i,t+1} = f_O(o_{i,t}, \sum_{k \in N_i} e_{k,t}), \quad i = 1, \ldots, |O|,$$

where $o_{i,t} = (x_{i,t}, a_i^t, p_{i,t})$ denotes object $i$ at time $t$, $u_k$ and $v_k$ are the receiver and sender of relation $r_k$, and $N_i$ 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 shown good performance 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 the Newton’s cradle shown in Fig. 2, where ball A’s impact produces a compression wave that propagates through the
Hence, inspired by the ideas on fast RNNs training [24], [25], we propose to encode the shared information beforehand and reuse them along the propagation steps. We denote the encoders for objects as \( f^\text{enc}_O \) and the encoder for relations as \( f^\text{enc}_R \). Then,

\[
\begin{align*}
    c^O_{i,t} &= f^\text{enc}_O(o_{i,t}), \\
    c^R_{k,t} &= f^\text{enc}_R(o_{u_k,t}, o_{v_k,t}, a^r_{k,t}).
\end{align*}
\]

In practice, we add residual links [26] between adjacent propagation steps that connect \( h^l_{i,t} \) and \( h^{l-1}_{i,t} \). This helps address the gradient vanishing/exploding problem and provides access to the historical effects. The update rules become

\[
\begin{align*}
    c^O_{i,t} &= f^\text{enc}_O(o_{i,t}), \\
    c^R_{k,t} &= f^\text{enc}_R(o_{u_k,t}, o_{v_k,t}, a^r_{k,t}), \\
    h^l_{i,t} &= f^I_R(c^O_{i,t}, h^l_{u_k,t}, h^{l-1}_{v_k,t}), \\
    h^l_{i,t} &= f^I_O(c^O_{i,t}, \sum_{k \in N_i} c^{l-1}_k, h^{l-1}_{i,t}),
\end{align*}
\]

where propagators \( f^I_O \) and \( f^I_R \) now take a new set 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 the Newton’s cradle), we can use small networks as function approximators for the propagators \( f^I_O \) and \( f^I_R \) for better efficiency. We name this updated model Propagation Networks (PropNets).

C. Partially Observable Scenarios

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

\[
\tau(G_{t+1}) = \phi(\tau(G_t)),
\]

where \( \tau \) is an encoding function that maps the current state to a latent representation. Depending on the actual scenarios, both \( \phi \) and \( \tau \) can be realized as PropNets. Note that in fully observable environments, \( \tau \) reduces to an identity mapping.

To train such a latent dynamics model, we seek to minimize the loss function: \( L_{\text{forward}} = ||\tau(G_{t+1}) - \phi(\tau(G_t))|| \). Using this 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 \( \tau \) should reserve information about the scene state. Hence, we introduce a decoding function \( \psi \) to ensure a nontrivial \( \tau \) by minimizing an additional auto-encoder reconstruction loss [27]: \( L_{\text{encode}} = ||G - \psi(\tau(G))|| \).

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 the simulated results and the goal. The model can also estimate the uncertain attributes online by minimizing the difference between the predicted future and the reality. Alg. [ ] outlines our control algorithm, which provides a good testbed for evaluating the modeling of the dynamics.

a) Model predictive control using shooting methods:
Let \( \mathcal{G}_g \) be our goal and \( \tilde{u}_{1:T} \) 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 \( \mathcal{G}_T = \tau(\mathcal{G}) \), and the resulting trajectory after applying the control inputs as \( \mathcal{G} = \{ \mathcal{G}_i \}_{i=1:T} \). The task here is to determine the control inputs by minimizing the gap between the actual outcome and the specified goal \( \mathcal{L}_g(\mathcal{G}, \mathcal{G}_g) \).

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, \( \mathcal{G}_{t+1} = \phi(\mathcal{G}_T) \). Let’s denote the forward simulation from time step \( t \) as \( \mathcal{G}_T \) and the history until time \( t \) as \( \mathcal{G}_t = \{ \mathcal{G}_i \}_{i=1...,t} \). We can back-propagate from the loss \( \mathcal{L}_g(\mathcal{G} \cup \mathcal{G}_g, \mathcal{G}_g) \) and use stochastic gradient descent (SGD) to update the control inputs. This is known as the shooting method in trajectory optimization [28].

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) [29] 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, without actually interacting with the objects, inherent attributes such as masses, friction, and damping are not directly observable. PropNet can estimate these attributes online (denoted as \( A \)) with SGD updates by minimizing the difference between the predicted future states and the actual future states \( \mathcal{L}_s(\mathcal{G}_T, \mathcal{G}_T) \).

### V. Experiments

In this section, we proceed to evaluate our method’s performance on both simulation and control in three scenarios: Newton’s Cradle, String Manipulation and Box Pushing. We also test how it generalizes and 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 [5]. For Vanilla PropNet, we adopt similar network structure where the relation propagator \( f^R_\theta(l \leq l \leq L) \) is an MLP with four 150-dim hidden layers and the object propagator \( f^O_\theta(l \leq l \leq L - 1) \) has one 100-dim hidden layer. Both output a 100-dim propagation vector. For fully observable scenarios, \( f^O_\theta \) has one 100-dim hidden layer and outputs a 2-dim vector representing the velocity at the next time step. For partially observable cases, \( f^O_\theta \) outputs one 100-dim vector as the latent representation.

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 setup, we assume full-state observation and the graph \( G \) of \( n \) balls has 2\( n \) objects representing the balls and the corresponding fixed pinpoints above the balls, as can be seen in Fig. 2a, where \( n = 5 \). There will be 2\( n \) directed relations describing the rigid connections between the fixed points and the balls. Collisions between adjacent balls introduce another 2\((n-1)\) relations.

For PropNet, we use an MLP with three 150-dim hidden layers as the relation encoder \( f^R_\theta^{enc} \) and one 100-dim hidden layer MLP as the object encoder \( f^O_\theta^{enc} \). Light-weight neural networks are used for the propagators \( f^R_\theta \) and \( f^O_\theta \), 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 setup, we assume full-state observation and the graph \( G \) of \( n \) balls has 2\( n \) objects representing the balls and the corresponding fixed pinpoints above the balls, as can be seen in Fig. 2a, where \( n = 5 \). There will be 2\( n \) 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 for 2,000 epochs with a mini-batch of 32. We use the Adam optimizer [30] with an initial learning rate of 0.001. We downscale 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 can not 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
Two circular obstacles are placed at random positions near a 15-dim string and evaluated in situations where the string described above. Fig. 4a and Fig. 5a show qualitative and modeled as a directed edge connecting the mass itself. another relation with each fixed obstacle, which adds to the graph in the dynamics graph \( G \) connecting each other, resulting in \( \text{of adjacent masses on the string will have spring relations} \) \( n \) particles, there will be a total of \( n + 2 \) objects. Each pair of adjacent masses on the string 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 applied to the masses on the string and the string is moving in compliant with the forces. We also include frictional forces in this scenario. More specifically, for a string containing \( n \) particles, there will be a total of \( n + 2 \) objects. Each pair of adjacent masses on the string 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 and training procedure as described above. Fig. 4a and Fig. 5a show qualitative and quantitative results, respectively. We train the models with a 15-dim string and evaluated in situations where the string length can vary between 10 and 20. As can be seen from the figures, although in this case, 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 with a large margin.

d) **Box pushing.**: In this case, we are pushing a pile of boxes forward (Fig. 4c). 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. We augment the encoding function \( \tau \) by averaging the object-centric outputs before feeding to \( \phi \). The dynamics function \( \phi \) 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. 5a and 5b show the ablation studies regarding the choice of \( L \). PropNet achieves a good accuracy at \( L = 3 \), which also has a good speed/accuracy trade-off. 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 as described in Section 3.B.
Adam optimizer. We compare our model with IN. Qualitative (e.g., mass, friction, damping) and compare the performance (Fig. 4b). The controller tries to match the target configuration where the ground-truth attributes are known (“Normal”), where the value (“Genearlize”). DRL has the same performance for “Bias” and “Adapt” as MSE optimized with PPO [32] - DRL), as well as Interaction free Deep Reinforcement Learning (Actor-Critic method between the resulting configuration and the goal configuration.

L dynamics of the string. The loss by “swinging” the string, which requires to leverage the controls are the top two masses at the moving end of the string to move the string to a target configuration, where the only state observation and a control task would be to determine the target configurations. The negative of the distance is used between the observable boxes at the end of the episode and the target scene encoding.

B. Control

We now evaluate the applicability of the learned model on control tasks. We first describe the three tasks: Newton’s Cradle, String 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 \( L_g(G, G_0) \). In this specific task, the loss \( L_g \) is the \( L_2 \) distance between the target height of the right-most ball and the highest height that has been achieved in \( G \).

We initialize the swing up angle as \(45^\circ\) 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. 2. 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) String Manipulation.: Here we define the task as to move the string to a target configuration, where the only controls are the top two masses at the moving end of the string (Fig 3a). The controller tries to match the target configuration by “swinging” the string, which requires to leverage the dynamics of the string. The loss \( L_g \) here is the \( L_2 \) 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) [31], Model-free Deep Reinforcement Learning (Actor-Critic method optimized with PPO [32] - DRL), as well as Interaction Networks (IN) and Propagation Networks (PropNet) with Alg. 1. Fig. 6 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. 4b 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 string.

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. 6 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. 6).

We further test whether our model generalizes to new scenarios, where the length of the string is varied between 10 to 20. As can be seen in Fig. 6, 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. 4c). 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 \( L_g \) used for MPC is the \( L_2 \) distance between the resulting scene encoding and the target scene encoding.

We evaluate the performance by the Chamfer Distance [33] between the observable boxes at the end of the episode and the target configurations. The negative of the distance is used as the reward for DRL. Fig. 4c and Fig. 6b 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 a long-range interactions within a single time step. We have also proposed to improve PropNet’s efficiency by adding residual connections and shared encoding.
