Dynamic Inference on Graphs using Structured Transition Models

Saumya Saxena[1] and Oliver Kroemer[1]

***Abstract*—Common everyday tasks such as picking up an object in one smooth motion or pushing off a wall to quickly turn a corner involve complex dynamic interactions between the human and the environment. These dynamic interactions are critical in successful execution of these tasks. Thus, in order to enable robots to perform such dynamic tasks effectively, we need to consider the dynamics of the robot, the individual objects, as well as the interactions between them. In this work, we present a method that enables efficient learning of the dynamics of interacting systems by simultaneously learning a *dynamic* graph structure and a *stable* and *locally linear* forward dynamic model of the system. The dynamic graph structure encodes evolving contact modes along a trajectory by making probabilistic predictions over the *edge activations*. The learned stable and locally linear dynamics enable the use of optimal control algorithms such as iLQR for long-horizon planning and control for complex interactive tasks. Through experiments in simulation and in the real world, we evaluate the performance of our method by using the learned interaction dynamics for control and demonstrate generalization to more objects and interactions not seen during training.**

I. Introduction

In this work, we develop a method for efficiently learning the dynamics of interacting systems by simultaneously learning a relational *dynamic* graph structure and a *stable locally linear* forward dynamic model of the system. We use the spring-mass-damper model as a structural prior over the *local* object-centric dynamics of our system. This provides us with two benefits: linearity and stability. For positive values of mass, stiffness and damping parameters, the linear spring-mass-damper model is always stable about the equilibrium point (which is also learned) [2].

We explore an exciting application of this work in the field of apprenticeship learning wherein the learned locally linear model can be used for learning the quadratic cost function underlying expert demonstrations using differentiable LQR as the policy class [1, 4]. The learned behavior can then be generalized to unseen goal conditions.

The key contributions of this work are two-fold: 1) a method for learning stable locally linear dynamics for non-linear interactive systems using graph neural networks by encoding changing dynamics and contacts as part of the graph structure, enabling strong generalization properties to more objects in the scene, and 2) using the learned locally linear dynamics to devise a robust control scheme that utilizes the recurrent nature of the learned graph structure to adapt the model predictions and the policy to observed contact events.

[1]Robotics Institute, School of Computer Science, Carnegie Mellon University, Pittsburgh PA 15123, USA {saumyas, okroemer}@andrew.cmu.edu

II. Method

Our training pipeline is composed of two main modules that are trained together, 1) a graph inference module that learns the dynamic graph structure, and 2) a forward dynamics module that learns the stable locally linear forward dynamics given the graph structure. An overview of the method is shown in Fig. 1.

A. Graph Inference Module

The state of the system at time t is represented as a graph $\mathcal{G}_t = (\mathcal{N}_t, \mathcal{E}_t)$ where nodes $\mathcal{N}_t$ represent objects in the scene and edges $\mathcal{E}_t$ connect interacting nodes. Node $n_t^i \in \mathcal{N}_t$ is represented using a feature vector $\mathbf{x}_t^i$ where $\mathbf{x}_t^i = [\mathbf{q}_t^i, \dot{\mathbf{q}}_t^i]$. $\mathbf{q}_t^i$ and $\dot{\mathbf{q}}_t^i$ are the position and velocity of the ith object respectively. The edge $e_t^{ij} \in \mathcal{E}_t$ connecting nodes *i* and *j* is represented with edge features $\mathbf{d}_t^{ij}$ where $\mathbf{d}_t^{ij}$ is the distance between nodes n_t^i and n_t^j.

Starting with a set of trajectories of fully connected graphs $(\mathcal{G}_0, \mathbf{u}_0, \ldots, \mathcal{G}_T, \mathbf{u}_T)$ (where $\mathbf{u}_t$ is the control applied at time t, and T is the length of the trajectory), we first pass them through the inference graph network that performs message passing between the nodes and outputs embedded graphs. Next, to aggregate information temporally, we pass the edge embeddings $\mathbf{m}_{0,\ldots,T}^{ij}$ through a GRU that outputs a discrete probability distribution $\mathbf{p}_t^{ij}$ over the edge types (active or inactive) for each edge. We sample from this distribution to get the edge activations $\mathbf{a}_t^{ij}$. Differentiability of this sampling procedure can be ensured by employing techniques such as Gumbel softmax [3].

B. Forward Dynamics Module

Using the edge activations output from the Graph Inference Module, we remove the inactive edges from the fully connected input graphs and then pass them through the forward dynamics graph network that outputs another set of embedded graphs. The node embeddings $\tilde{\mathbf{z}}_t^{ij}$ are passed though a fully connected neural network that outputs spring-mass-damper parameters $[\frac{1}{m_t^i}, k_t^i, c_t^i, \hat{x}_t^i]$ independently for each node where m_t^i is the mass, k_t^i is the stiffness of the spring, c_t^i is the damping, $\hat{x}_t^i$ is the equilibrium point.

The loss function for training the model is written as,

$$\mathcal{L}_{\text{dyn}} = \sum_i ||\mathbf{x}_{1:T}^{i^*} - \mathbf{x}_{1:T}^{i^{\text{pred}}}||_2 + \sum_{i,j} \text{KL}[\mathbf{p}_{0:T-1}^{ij} \,||\, \mathbf{q}_{0:T-1}^{ij}]$$

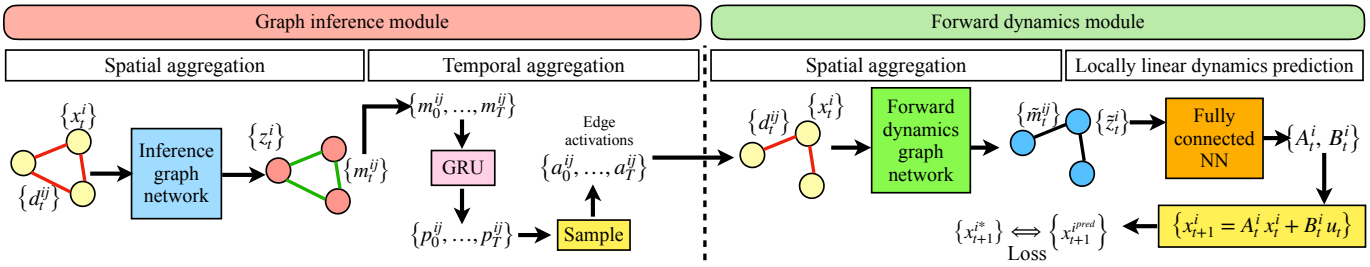

Fig. 1. Method overview. Our method has two main parts 1) Graph inference module: we start with a set of fully connected graphs and pass them through the inference graph network that performs message passing between the nodes and outputs embedded graphs. Next, to aggregate information temporally, we pass the edge embeddings $\mathbf{m}^{ij}_{0,\ldots,T}$ through a GRU that outputs a discrete probability distribution $\mathbf{p}^{ij}_t$ over the edge types (active or inactive) for each edge. We sample from this distribution to get the edge activations $\mathbf{a}^{ij}_t$. 2) Forward dynamics module: using the edge activations, we remove the inactive edges from the fully connected input graphs and then pass them through the forward dynamics graph network that outputs another set of embedded graphs. The node embeddings $\tilde{\mathbf{z}}^{ij}_t$ are passed though a fully connected neural network that outputs stable locally linear transition dynamics. We forward propagate the learned dynamics using the input control and get the next state which is used for loss calculation during training

where $\mathbf{x}^{i*}_{t+1}$ is the observed next state and $\mathbf{x}^{i^{\text{pred}}}_{t+1} = \mathbf{A}^i_t \mathbf{x}^{i^*}_t + \mathbf{B}^i_t \mathbf{u}_t + \mathbf{o}^i_t$ is the predicted next state for node n^i. $\mathbf{p}^{ij}_{0:T-1}$ is the probability distribution over predicted edge activations and $\mathbf{q}^{ij}_{0:T-1}$ is a prior on the edge activations.

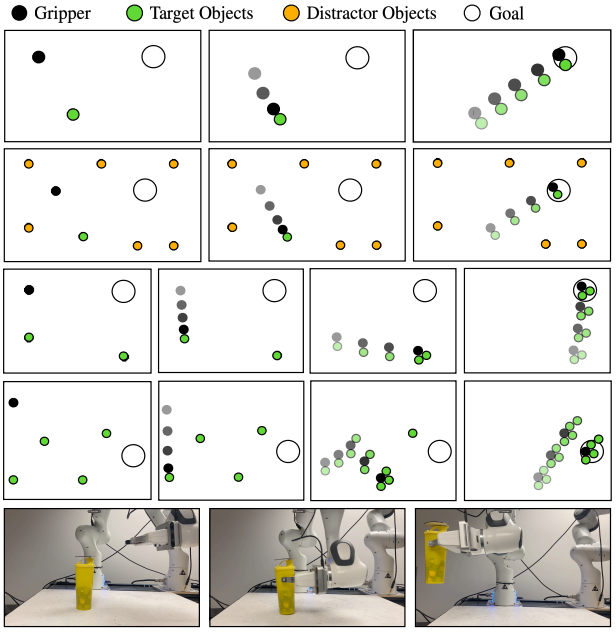

Fig. 2. Box2D and real world experiments

III. EXPERIMENTS

The proposed method is evaluated in simulation using the Box2D environment as shown in Fig. 2. We consider two tasks in this environment: **Task 1:** Dynamically (without stopping) picking up a 2D object using a 2D gripper and taking it to a goal region (Fig. 2 Rows 1 & 2), and **Task 2:** Dynamic pickup of multiple objects (Fig. 2 Rows 3 & 4). For each of the tasks, we use the learned dynamic models for control using iLQR-MPC and calculate the N-step prediction error. Row 1 in Fig. 2 shows the executed trajectory for Box2D Task 1 with same number of objects as seen during training. Row 2 shows Box2D Task 1 generalization scenario with many distractor objects (orange) in the scene, not seen during training. Row 3 shows Box2D Task 2 with same number of objects (two) as seen during training. Row 4 shows Box2D Task 2 generalization scenario for picking up many more objects than during training. Row 5 shows real world experiments with the 7DOF Franka-Emika Panda arm performing a dynamic pickup task using the learned model.

We observe that, when tested with the same number of object interactions as seen during training, the N-step prediction error is 0.16cm for task 1 and 0.14cm for task 2. When tested for generalization (rows 2 and 4 in Fig. 2) the N-step prediction error is 0.34cm for task 1 and 0.39cm for task 2 – less than a centimeter. Hence, we show strong generalization to more objects and interactions in the scene.

IV. CONCLUSION AND FUTURE WORK

In this work, we showed that by learning the dynamics of interactive systems by simultaneously learning a dynamic interaction graph and a *stable* locally linear forward dynamic model given the graph, we can use the learned dynamics for long-horizon control using iLQR-MPC. This work also demonstrates on the generalization benefits of learning a dynamic graph structure.

REFERENCES

[1] Brandon Amos, Ivan Jimenez, Jacob Sacks, Byron Boots, and J Zico Kolter. Differentiable mpc for end-to-end planning and control. *Advances in neural information processing systems*, 31, 2018.

[2] Karl Johan Åström and Richard M Murray. *Feedback systems, Chapter 8*. Princeton university press, 2010.

[3] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[4] Saumya Saxena, Alex LaGrassa, and Oliver Kroemer. Learning reactive and predictive differentiable controllers for switching linear dynamical models. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7563–7569. IEEE, 2021.