Graph Networks as Learnable Physics Engines for Inference and Control

Alvaro Sanchez-Gonzalez [1] **Nicolas Heess** [1] **Jost Tobias Springenberg** [1] **Josh Merel** [1] **Martin Riedmiller** [1]
Raia Hadsell [1] **Peter Battaglia** [1]

Abstract

Understanding and interacting with everyday physical scenes requires rich knowledge about the structure of the world, represented either implicitly in a value or policy function, or explicitly in a transition model. Here we introduce a new class of learnable models—based on *graph networks*—which implement an inductive bias for object- and relation-centric representations of complex, dynamical systems. Our results show that as a forward model, our approach supports accurate predictions from real and simulated data, and surprisingly strong and efficient generalization, across eight distinct physical systems which we varied parametrically and structurally. We also found that our inference model can perform system identification. Our models are also differentiable, and support online planning via gradient-based trajectory optimization, as well as offline policy optimization. Our framework offers new opportunities for harnessing and exploiting rich knowledge about the world, and takes a key step toward building machines with more human-like representations of the world.

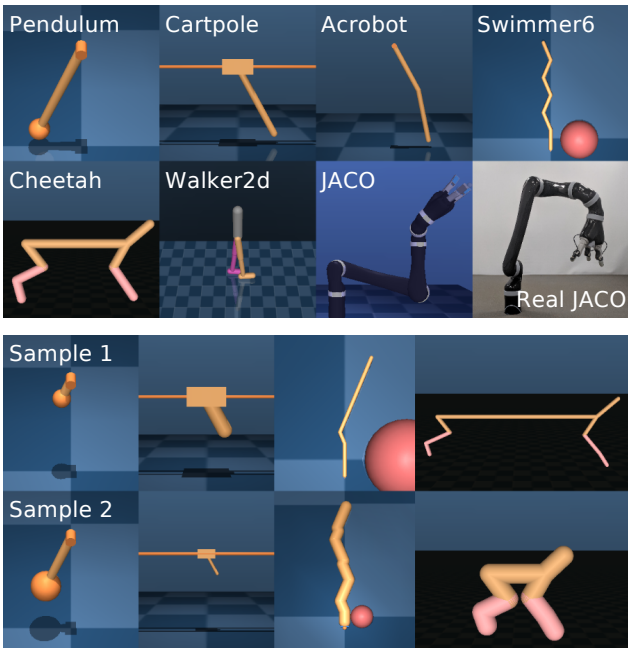

Figure 1. (Top) Our experimental physical systems. (Bottom) Samples of parametrized versions of these systems (see videos: link).

1. Introduction

Many domains, such as mathematics, language, and physical systems, are combinatorially complex. The possibilities scale rapidly with the number of elements. For example, a multi-link chain can assume shapes that are exponential in the number of angles each link can take, and a box full of bouncing balls yields trajectories which are exponential in the number of bounces that occur. How can an intelligent agent understand and control such complex systems?

A powerful approach is to represent these systems in terms of objects[2] and their relations, applying the same object-wise computations to all objects, and the same relation-wise computations to all interactions. This allows for combinatorial generalization to scenarios never before experienced, whose underlying components and compositional rules are well-understood. For example, particle-based physics engines make the assumption that bodies follow the same dynamics, and interact with each other following similar rules, e.g., via forces, which is how they can simulate limitless scenarios given different initial conditions.

Here we introduce a new approach for learning and controlling complex systems, by implementing a structural inductive bias for object- and relation-centric representations. Our approach uses "graph networks" (GNs), a class of neural networks that can learn functions on graphs (Scarselli et al., 2009b; Li et al., 2015; Battaglia et al., 2016; Gilmer et al., 2017). In a physical system, the GN lets us represent

[1]DeepMind, London, United Kingdom. Correspondence to: Alvaro Sanchez-Gonzalez <alvarosg@google.com>, Peter Battaglia <peterbattaglia@google.com>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

[2]"Object" here refers to entities generally, rather than physical objects exclusively.

the bodies (objects) with the graph's nodes and the joints (relations) with its edges. During learning, knowledge about body dynamics is encoded in the GN's node update function, interaction dynamics are encoded in the edge update function, and global system properties are encoded in the global update function. Learned knowledge is shared across the elements of the system, which supports generalization to new systems composed of the same types of body and joint building blocks.

Across seven complex, simulated physical systems, and one real robotic system (see Figure 1), our experimental results show that our GN-based *forward models* support accurate and generalizable predictions, *inference models*[3] support system identification in which hidden properties are abduced from observations, and *control algorithms* yield competitive performance against strong baselines. This work represents the first general-purpose, learnable physics engine that can handle complex, 3D physical systems. Unlike classic physics engines, our model has no specific a priori knowledge of physical laws, but instead leverages its object- and relation-centric inductive bias to learn to approximate them via supervised training on current-state/next-state pairs.

Our work makes three technical contributions: GN-based forward models, inference models, and control algorithms. The forward and inference models are based on treating physical systems as graphs and learning about them using GNs. Our control algorithm uses our forward and inference models for planning and policy learning.

(For full algorithm, implementation, and methodological details, as well as videos from all of our experiments, please see the Supplementary Material.)

2. Related Work

Our work draws on several lines of previous research. Cognitive scientists have long pointed to rich generative models as central to perception, reasoning, and decision-making (Craik, 1967; Johnson-Laird, 1980; Miall & Wolpert, 1996; Spelke & Kinzler, 2007; Battaglia et al., 2013). Our core model implementation is based on the broader class of graph neural networks (GNNs) (Scarselli et al., 2005; 2009a;b; Bruna et al., 2013; Li et al., 2015; Henaff et al., 2015; Duvenaud et al., 2015; Dai et al., 2016; Defferrard et al., 2016; Niepert et al., 2016; Kipf & Welling, 2016; Battaglia et al., 2016; Watters et al., 2017; Raposo et al., 2017; Santoro et al., 2017; Bronstein et al., 2017; Gilmer et al., 2017). One of our key contributions is an approach for learning physical dynamics models (Grzeszczuk et al., 1998; Fragkiadaki et al., 2015; Battaglia et al., 2016; Chang et al., 2016; Watters et al., 2017; Ehrhardt et al., 2017; Amos et al., 2018). Our inference model shares similar aims as approaches for learning system identification explicitly (Yu et al., 2017; Peng et al., 2017), learning policies that are robust to hidden property variations (Rajeswaran et al., 2016), and learning exploration strategies in uncertain settings (Schmidhuber, 1991; Sun et al., 2011; Houthooft et al., 2016). We use our learned models for model-based planning in a similar spirit to classic approaches which use pre-defined models (Li & Todorov, 2004; Tassa et al., 2008; 2014), and our work also relates to learning-based approaches for model-based control (Atkeson & Santamaria, 1997; Deisenroth & Rasmussen, 2011; Levine & Abbeel, 2014). We also explore jointly learning a model and policy (Heess et al., 2015; Gu et al., 2016; Nagabandi et al., 2017). Notable recent, concurrent work (Wang et al., 2018) used a GNN to approximate a policy, which complements our use of a related architecture to approximate forward and inference models.

3. Model

Graph representation of a physical system. Our approach is founded on the idea of representing physical systems as graphs: the bodies and joints correspond to the nodes and edges, respectively, as depicted in Figure 2a. Here a (directed) *graph* is defined as $G = (\mathbf{g}, \{\mathbf{n}_i\}_{i=1\cdots N_n}, \{\mathbf{e}_j, s_j, r_j\}_{j=1\cdots N_e})$, where $\mathbf{g}$ is a vector of global features, $\{\mathbf{n}_i\}_{i=1\cdots N_n}$ is a set of nodes where each $\mathbf{n}_i$ is a vector of node features, and $\{\mathbf{e}_j, s_j, r_j\}_{j=1\cdots N_e}$ is a set of directed edges where $\mathbf{e}_j$ is a vector of edge features, and s_j and r_j are the indices of the sender and receiver nodes, respectively.

We distinguish between static and dynamic properties in a physical scene, which we represent in separate graphs. A static graph G_s contains static information about the parameters of the system, including global parameters (such as the time step, viscosity, gravity, etc.), per body/node parameters (such as mass, inertia tensor, etc.), and per joint/edge parameters (such as joint type and properties, motor type and properties, etc.). A dynamic graph G_d contains information about the instantaneous state of the system. This includes each body/node's 3D Cartesian position, 4D quaternion orientation, 3D linear velocity, and 3D angular velocity.[4] Additionally, it contains the magnitude of the actions applied to the different joints in the corresponding edges.

[3] We use the term "inference" in the sense of "abductive inference"—roughly, constructing explanations for (possibly partial) observations—and not probabilistic inference, per se.

[4] Some physics engines, such as Mujoco (Todorov et al., 2012), represent systems using "generalized coordinates", which sparsely encode degrees of freedom rather than full body states. Generalized coordinates offer advantages such as preventing bodies connected by joints from dislocating (because there is no degree of freedom for such displacement). In our approach, however, such representations do not admit sharing as naturally because there are different input and output representations for a body depending on the system's constraints.

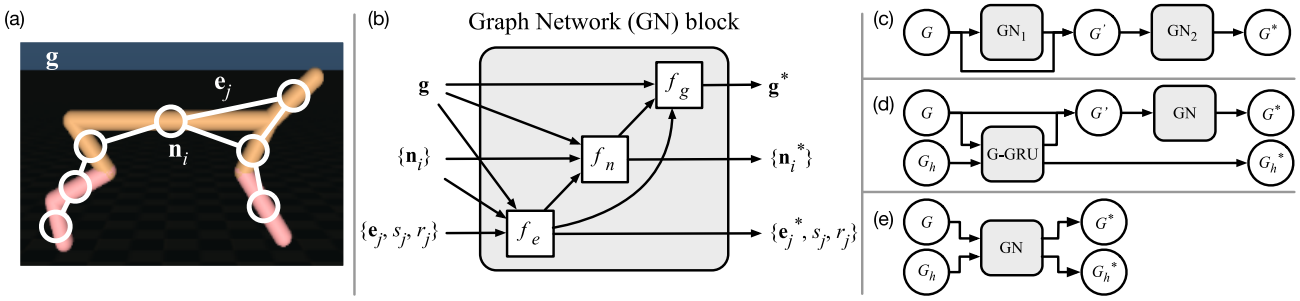

Figure 2. Graph representations and GN-based models. (a) A physical system's bodies and joints can be represented by a graph's nodes and edges, respectively. (b) A GN block takes a graph as input and returns a graph with the same structure but different edge, node, and global features as output (see Algorithm 1). (c) A feed-forward GN-based forward model for learning one-step predictions. (d) A recurrent GN-based forward model. (e) A recurrent GN-based inference model for system identification.

Algorithm 1 Graph network, GN

Input: Graph, $G = (\mathbf{g}, \{\mathbf{n}_i\}, \{\mathbf{e}_j, s_j, r_j\})$
for each edge $\{\mathbf{e}_j, s_j, r_j\}$ **do**
  Gather sender and receiver nodes $\mathbf{n}_{s_j}, \mathbf{n}_{r_j}$
  Compute output edges, $\mathbf{e}_j^* = f_e(\mathbf{g}, \mathbf{n}_{s_j}, \mathbf{n}_{r_j}, \mathbf{e}_j)$
end for
for each node $\{\mathbf{n}_i\}$ **do**
  Aggregate $\mathbf{e}_j^*$ per receiver, $\hat{\mathbf{e}}_i = \sum_{j/r_j=i} \mathbf{e}_j^*$
  Compute node-wise features, $\mathbf{n}_i^* = f_n(\mathbf{g}, \mathbf{n}_i, \hat{\mathbf{e}}_i)$
end for
Aggregate all edges and nodes $\hat{\mathbf{e}} = \sum_j \mathbf{e}_j^*$, $\hat{\mathbf{n}} = \sum_i \mathbf{n}_i^*$
Compute global features, $\mathbf{g}^* = f_g(\mathbf{g}, \hat{\mathbf{n}}, \hat{\mathbf{e}})$
Output: Graph, $G^* = (\mathbf{g}^*, \{\mathbf{n}_i^*\}, \{\mathbf{e}_j^*, s_j, r_j\})$

Graph networks. The GN architectures introduced here generalize interaction networks (IN) (Battaglia et al., 2016) in several ways. They include global representations and outputs for the state of a system, as well as per-edge outputs. They are defined as "graph2graph" modules (i.e., they map input graphs to output graphs with different edge, node, and global features), which can be composed in deep and recurrent neural network (RNN) configurations. A core GN block (Figure 2b) contains three sub-functions—edge-wise, f_e, node-wise, f_n, and global, f_g—which can be implemented using standard neural networks. Here we use multi-layer perceptrons (MLP). A single feedforward GN pass can be viewed as one step of message-passing on a graph (Gilmer et al., 2017), where f_e is first applied to update all edges, f_n is then applied to update all nodes, and f_g is finally applied to update the global feature. See Algorithm 1 for details.

Forward models. For prediction, we introduce a GN-based forward model for learning to predict future states from current ones. It operates on one time-step, and contains two GNs composed sequentially in a "deep" arrangement (unshared parameters; see Figure 2c). The first GN takes an input graph, G, and produces a latent graph, G'. This G' is concatenated[5] with G (e.g., a graph skip connection), and provided as input to the second GN, which returns an output graph, G^*. Our forward model training optimizes the GN so that G^*'s $\{\mathbf{n}_i\}$ features reflect predictions about the states of each body across a time-step. The reason we used two GNs was to allow all nodes and edges to communicate with each other through the $\mathbf{g}'$ output from the first GN. Preliminary tests suggested this provided large performance advantages over a single IN/GN (see ablation study in SM Figure H.2).

We also introduce a second, recurrent GN-based forward model, which contains three RNN sub-modules (GRUs, (Cho et al., 2014)) applied across all edges, nodes, and global features, respectively, before being composed with a GN block (see Figure 2d).

Our forward models were all trained to predict state differences, so to compute absolute state predictions we updated the input state with the predicted state difference. To generate a long-range *rollout* trajectory, we repeatedly fed absolute state predictions and externally specified control inputs back into the model as input, iteratively. As data pre- and post-processing steps, we normalized the inputs and outputs to the GN model.

Inference models. System identification refers to inferences about unobserved properties of a dynamic system based on its observed behavior. It is important for controlling systems whose unobserved properties influence the control dynamics. Here we consider "implicit" system identification, in which inferences about unobserved properties are not estimated explicitly, but are expressed in latent representations which are made available to other mechanisms.

We introduce a recurrent GN-based inference model, which observes only the dynamic states of a trajectory and con-

[5]We define the term "graph-concatenation" as combining two graphs by concatenating their respective edge, node, and global features. We define "graph-splitting" as splitting the edge, node, and global features of one graph to form two new graphs with the same structure.

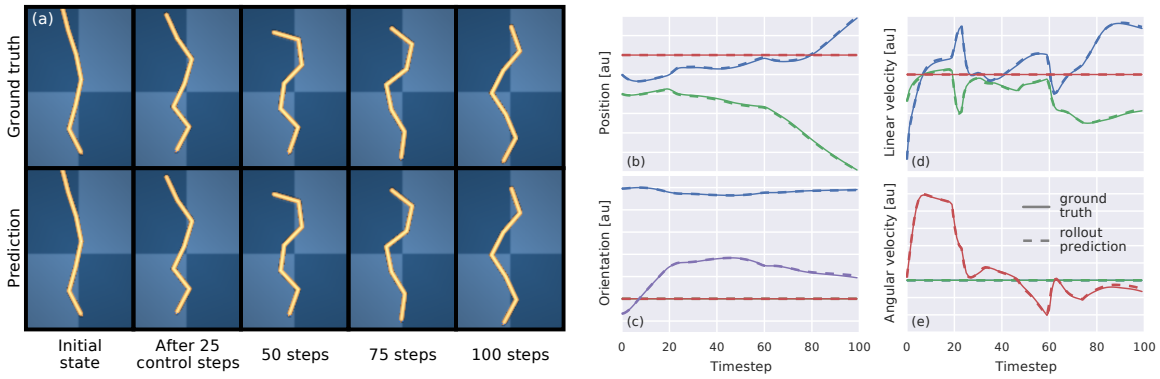

Figure 3. Evaluation rollout in a Swimmer6. Trajectory videos are here: link-P.F.S6. (a) Frames of ground truth and predicted states over a 100 step trajectory. (b-e) State sequence predictions for link #3 of the Swimmer. The subplots are (b) x, y, z-position, (c) $q0, q1, q2, q3$-quaternion orientation, (d) x, y, z-linear velocity, and (e) x, y, z-angular velocity. [au] indicates arbitrary units.

structs a latent representation of the unobserved, static properties (i.e., performs implicit system identification). It takes as input a sequence of dynamic state graphs, G_d, under some control inputs, and returns an output, $G^*(T)$, after T time steps. This $G^*(T)$ is then passed to a one-step forward model by graph-concatenating it with an input dynamic graph, G_d. The recurrent core takes as input, G_d, and hidden graph, G_h, which are graph-concatenated[5] and passed to a GN block (see Figure 2e). The graph returned by the GN block is graph-split[5] to form an output, G^*, and updated hidden graph, G_h^*. The full architecture can be trained jointly, and learns to infer unobserved properties of the system from how the system's observed features behave, and use them to make more accurate predictions.

Control algorithms. For control, we exploit the fact that the GN is differentiable to use our learned forward and inference models for model-based planning within a classic, gradient-based trajectory optimization regime, also known as model-predictive control (MPC). We also develop an agent which simultaneously learns a GN-based model and policy function via Stochastic Value Gradients (SVG) (Heess et al., 2015). [6]

4. Methods

Environments. Our experiments involved seven actuated Mujoco simulation environments (Figure 1). Six were from the "DeepMind Control Suite" (Tassa et al., 2018)—Pendulum, Cartpole, Acrobot, Swimmer, Cheetah, Walker2d—and one was a model of a JACO commercial robotic arm. We generated training data for our forward models by applying simulated random controls to the systems, and recording the state transitions. We also trained models from recorded trajectories of a real JACO robotic under human control during a stacking task.

In experiments that examined generalization and system identification, we created a dataset of versions of several of our systems—Pendulum, Cartpole, Swimmer, Cheetah and JACO— with procedurally varied parameters and structure. We varied continuous properties such as link lengths, body masses, and motor gears. In addition, we also varied the number of links in the Swimmer's structure, from 3-15 (we refer to a swimmer with N links as Swimmer$N$).

MPC planning. We used our GN-based forward model to implement MPC planning by maximizing a dynamic-state-dependent reward along a trajectory from a given initial state. We used our GN forward model to predict the N-step trajectories (N is the planning *horizon*) induced by proposed action sequences, as well as the total reward associated with the trajectory. We optimized these action sequences by backpropagating gradients of the total reward with respect to the actions, and minimizing the negative reward by gradient descent, iteratively.

Model-based reinforcement learning. To investigate whether our GN-based model can benefit reinforcement learning (RL) algorithms, we used our model within an SVG regime (Heess et al., 2015). The GN forward model was used as a differentiable environment simulator to obtain a gradient of the expected return (predicted based on the next state generated by a GN) with respect to a parameterized, stochastic policy, which was trained jointly with the GN. For our experiments we used a single step prediction (SVG(1)) and compared to sample-efficient model-free RL baselines using either stochastic policies (SVG(0)) or deterministic policies via the Deep Deterministic Policy Gradients (DDPG) algorithm (Lillicrap et al., 2016) (which is also used as a baseline in the MPC experiments).

[6] MPC and SVG are deeply connected: in MPC the control inputs are optimized given the initial conditions in a single episode, while in SVG a policy function that maps states to controls is optimized over states experienced during training.

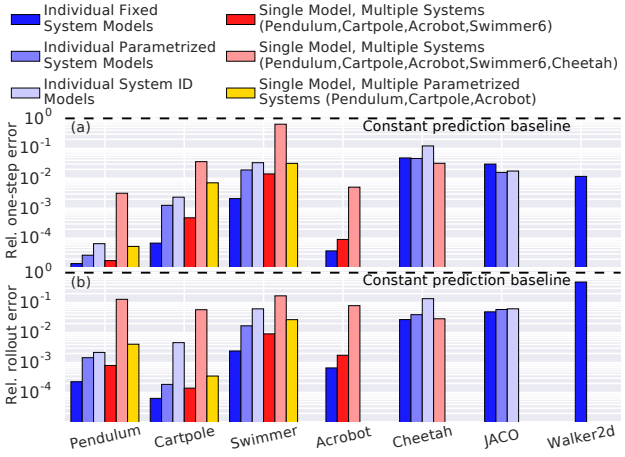

Figure 4. (a) One-step and (b) 100-step rollout errors for different models and training (different bars) on different test data (x-axis labels), relative to the constant prediction baseline (black dashed line). Blue bars are GN models trained on single systems. Red and yellow bars are GN models trained on multiple systems, with (yellow) and without (red) parametric variation. Note that including Cheetah in multiple system training caused performance to diminish (light red vs dark red bars), which suggests sharing might not always be beneficial.

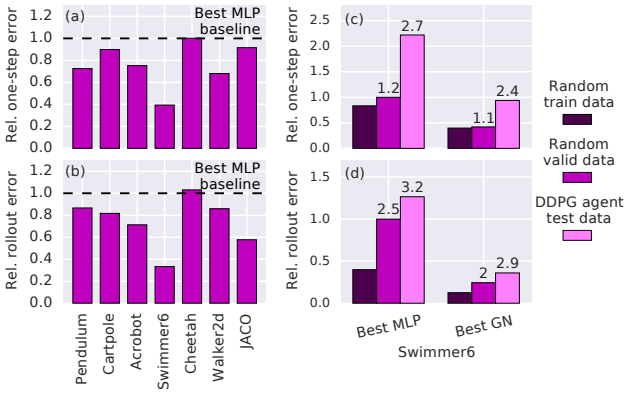

Figure 5. Prediction errors, on (a) one-step and (b) 20-step evaluations, between the best MLP baseline and the best GN model after 72 hours of training. Swimmer6 prediction errors, on (c) one-step and (d) 20-step evaluations, between the best MLP baseline and the best GN model for data in the training set (dark), data in the validation set (medium), and data from DDPG agent trajectories (light). The numbers above the bars indicate the ratio between the corresponding generalization test error (medium or light) and the training error (dark).

Baseline comparisons. As a simple baseline, we compared our forward models' predictions to a *constant prediction baseline*, which copied the input state as the output state. We also compared our GN-based forward model with a learned, MLP baseline, which we trained to make forward predictions using the same data as the GN model. We replaced the core GN with an MLP, and flattened and concatenated the graph-structured GN input and target data into a vector suitable for input to the MLP. We swept over 20 unique hyperparameter combinations for the MLP architecture, with up to 9 hidden layers and 512 hidden nodes per layer.

As an MPC baseline, with a pre-specified physical model, we used a Differential Dynamic Programming algorithm (Tassa et al., 2008; 2014) that had access to the ground-truth Mujoco model. We also used the two model-free RL agents mentioned above, SVG(0) and DDPG, as baselines in some tests. Some of the trajectories from a DDPG agent in Swimmer6 were also used to evaluate generalization of the forward models.

Prediction performance evaluation. Unless otherwise specified, we evaluated our models on squared one-step dynamic state differences (*one-step error*) and squared trajectory differences (*rollout error*) between the prediction and the ground truth. We calculated independent errors for position, orientation, linear velocity angular velocity, and normalized them individually to the constant prediction baseline. After normalization, the errors were averaged together. All errors reported are calculated for 1000 100-step sequences from the test set.

5. Results: Prediction

Learning a forward model for a single system. Our results show that the GN-based model can be trained to make very accurate forward predictions under random control. For example, the ground truth and model-predicted trajectories for Swimmer6 were both visually and quantitatively indistinguishable (see Figure 3). Figure 4's black bars show that the predictions across most other systems were far better than the constant prediction baseline. As a stronger baseline comparison, Figures 5a-b show that our GN model had lower error than the MLP-based model in 6 of the 7 simulated control systems we tested. This was especially pronounced for systems with much repeated structure, such as the Swimmer, while for systems with little repeated structure, such as Pendulum, there was negligible difference between the GN and MLP baseline. These results suggest that a GN-based forward model is very effective at learning predictive dynamics in a diverse range of complex physical systems.

We also found that the GN generalized better than the MLP baseline from training to test data, as well as across different action distributions. Figures 5c-d show that for Swimmer6, the relative increase in error from training to test data, and to data recorded from a learned DDPG agent, was smaller for the GN model than for the MLP baseline. We speculate that the GN's superior generalization is a result of implicit

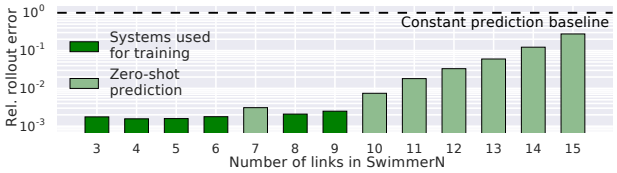

Figure 6. Zero-shot dynamics prediction. The bars show the 100-step rollout error of a model trained on a mixture of 3-6 and 8-9 link Swimmers, and tested on Swimmers with 3-15 links. The dark bars indicate test Swimmers whose number of links the model was trained on (video: link-P.F.SN), the light bars indicate Swimmers it was not trained on (video: link-P.F.SN(Z)).

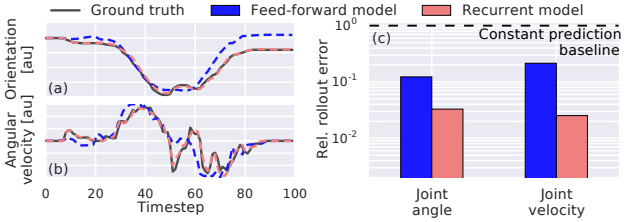

Figure 7. Real and predicted test trajectories of a JACO robot arm. The recurrent model tracks the ground truth (a) orientations and (b) angular velocities closely. (c) The total 100-step rollout error was much better for the recurrent model, though the feed-forward model was still well below the constant prediction baseline. A video of a Mujoco rendering of the true and predicted trajectories: link-P.F.JR.

regularization due to its inductive bias for sharing parameters across all bodies and joints; the MLP, in principle, could devote disjoint subsets of its computations to each body and joint, which might impair generalization.

Learning a forward model for multiple systems. Another important feature of our GN model is that it is very flexible, able to handle wide variation across a system's properties, and across systems with different structure. We tested how it learned forward dynamics of systems with continuously varying static parameters, using a new dataset where the underlying systems' bodies and joints had different masses, body lengths, joint angles, etc. These static state features were provided to the model via the input graphs' node and edge attributes. Figure 4 shows that the GN model's forward predictions were again accurate, which suggests it can learn well even when the underlying system properties vary.

We next explored the GN's inductive bias for body- and joint-centric learning by testing whether a single model can make predictions across multiple systems that vary in their number of bodies and the joint structure. Figure 6 shows that when trained on a mixed dataset of Swimmers with 3-6, 8-9 links, the GN model again learned to make accurate forward predictions. We pushed this even further by training a single GN model on multiple systems, with completely different structures, and found similarly positive results (see Figure 4, red and yellow bars). This highlights a key difference, in terms of general applicability, between GN and MLP models: the GN can naturally operate on variably structured inputs, while the MLP requires fixed-size inputs.

The GN model can even generalize, zero-shot, to systems whose structure was held out during training, as long as they are composed of bodies and joints similar to those seen during training. For the GN model trained on Swimmers with 3-6, 8-9 links, we tested on held-out Swimmers with 7 and 10-15 links. Figure 6 shows that zero-shot generalization performance is very accurate for 7 and 10 link Swimmers, and degrades gradually from 11-15 links. Still, their trajectories are visually very close to the ground truth (video: link-P.F.SN(Z)).

Real robot data. To evaluate our approach's applicability to the real world, we trained GN-based forward models on real JACO proprioceptive data; under manual control by a human performing a stacking task. We found the feed-forward GN performance was not as accurate as the recurrent GN forward model[7]: Figure 7 shows a representative predicted trajectory from the test set, as well as overall performance. These results suggest that our GN-based forward model is a promising approach for learning models in real systems.

6. Results: Inference

In many real-world settings the system's state is partially observable. Robot arms often use joint angle and velocity sensors, but other properties such as mass, joint stiffness, etc. are often not directly measurable. We applied our system identification inference model (see Model Section 3) to a setting where only the dynamic state variables (i.e., position, orientation, and linear and angular velocities) were observed, and found it could support accurate forward predictions (during its "prediction phase") after observing randomly controlled system dynamics during an initial 20-step "ID phase" (see Figure 8).

To further explore the role of our GN-based system identification, we contrasted the model's predictions after an ID phase, which contained useful control inputs, against an ID phase that did not apply control inputs, across three different Pendulum systems with variable, unobserved lengths. Figure 9 shows that the GN forward model with an identifiable ID phase makes very accurate predictions, but with an unidentifiable ID phase its predictions are very poor.

[7]This might result from lag or hysteresis which induces long-range temporal dependencies that the feed-forward model cannot capture.

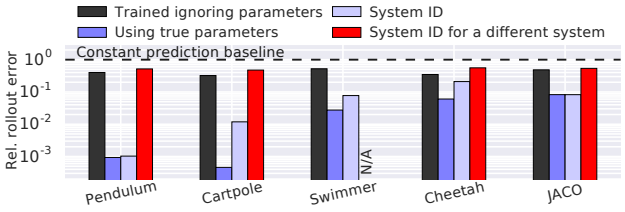

Figure 8. System identification performance. The y-axis represents 100-step rollout error, relative to the trivial constant prediction baseline (black dashed line). The baseline GN-based model (black bars) with no system identification module performs worst. A model which was always provided the true static parameters (medium blue bars) and thus did not require system identification performed best. A model without explicit access to the true static parameters, but with a system identification module (light blue bars), performed generally well, sometimes very close to the model which observed the true parameters. But when that same model was presented with an ID phase whose hidden parameters were different (but from the same distribution) from its prediction phase (red bars), its performance was similar or worse than the model (black) with no ID information available. (The N/A column is because our Swimmer experiments always varied the number of links as well as parameters, which meant the inferred static graph could not be concatenated with the initial dynamic graph.)

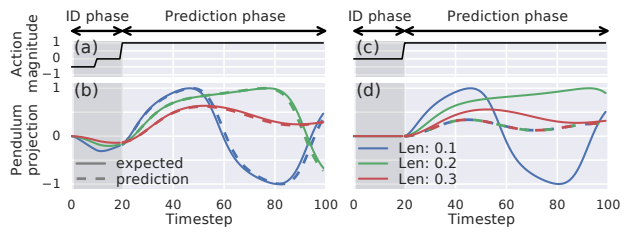

Figure 9. System identification analysis in Pendulum. (a) Control inputs are applied to three Pendulums with different, unobservable lengths during the 20-step ID phase, which makes the system identifiable. (b) The model's predicted trajectories (dashed curves) track the ground truth (solid curves) closely in the subsequent 80-step prediction phase. (c) No control inputs are applied to the same systems during the ID phase, which makes the system identifiable. (d) The model's predicted trajectories across systems are very different from the ground truth.

A key advantage of our system ID approach is that once the ID phase has been performed for some system, the inferred representation can be stored and reused to make trajectory predictions from different initial states of the system. This contrasts with an approach that would use an RNN to both infer the system properties and use them throughout the trajectory, which thus would require identifying the same system from data each time a new trajectory needs to be predicted given different initial conditions.

7. Results: Control

Differentiable models can be valuable for model-based sequential decision-making, and here we explored two approaches for exploiting our GN model in continuous control.

Figure 10. Frames from a 40-step GN-based MPC trajectory of the simulated JACO arm. (a) Imitation of the pose of each individual body of the arm (13 variables x 9 bodies). (b) Imitation of only the palm's pose (13 variables). The full videos are here: link-C.F.JA(o) and link-C.F.JA(a).

Model-predictive control for single systems. We trained a GN forward model and used it for MPC by optimizing the control inputs via gradient descent to maximize predicted reward under a known reward function. We found our GN-based MPC could support planning in all of our control systems, across a range of reward functions. For example, Figure 10 shows frames of simulated JACO trajectories matching a target pose and target palm location, respectively, under MPC with a 20-step planning horizon.

In the Swimmer6 system with a reward function that maximized the head's movement toward a randomly chosen target, GN-based MPC with a 100-step planning horizon selected control inputs that resulted in coordinated, swimming-like movements. Despite the fact that the Swimmer6 GN model used for MPC was trained to make one-step predictions under random actions, its swimming performance was close to both that of a more sophisticated planning algorithm which used the true Mujoco physics as its model, as well as that of a learned DDPG agent trained on the system (see Figure 11a). And when we trained the GN model using a mixture of both random actions and DDPG agent trajectories, there was effectively no difference in performance between our approach, versus the Mujoco planner and learned DDPG agent baselines (see video: link-C.F.S6).

For Cheetah with reward functions for maximizing forward movement, maximizing height, maximizing squared vertical speed, and maximizing squared angular speed of the torso, MPC with a 20-step horizon using a GN model resulted in running, jumping, and other reasonable patterns of movements (see video: link-C.F.Ch(k)).

Model-predictive control for multiple systems. Similar to how our forward models learned accurate predictions

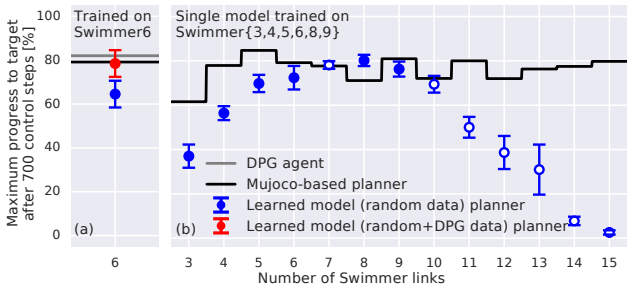

Figure 11. GN-based MPC performance (% distance to target after 700 steps) for (a) model trained on Swimmer6 and (b) model trained on Swimmers with 3-15 links (see Figure 6). In (a), GN-based MPC (blue point) is almost as good as the Mujoco-based planner (black line) and trained DDPG (grey line) baselines. When the GN-based MPC's model is trained on a mixture of random and DDPG agent Swimmer6 trajectories (red point), its performance is as good as the strong baselines. In (b) the GN-based MPC (blue point) (video: link-C.F.SN) is competitive with a Mujoco-based planner baseline (black) (video: link-C.F.SN(b)) for 6-10 links, but is worse for 3-5 and 11-15 links. Note, the model was not trained on the open points, 7 and 10-15 links, which correspond to zero-shot model generalization for control. Error bars indicate mean and standard deviation across 5 experimental runs.

across multiple systems, we also found they could support MPC across multiple systems (in this video, a single model is used for MPC in Pendulum, Cartpole, Acrobot, Swimmer6 and Cheetah: link-C.F.MS). We also found GN-based MPC could support zero-shot generalization in the control setting, for a single GN model trained on Swimmers with 3-6, 8-9 links, and tested on MPC on Swimmers with 7, 10-15 links. Figure 11b shows that it performed almost as well as the Mujoco baseline for many of the Swimmers.

Model-predictive control with partial observations. Because real-world control settings are often partially observable, we used the system identification GN model (see Sections 3 and 5) for MPC under partial observations in Pendulum, Cartpole, SwimmerN, Cheetah, and JACO. The model was trained as in the forward prediction experiments, with an ID phase that applied 20 random control inputs to implicitly infer the hidden properties. Our results show that our GN-based forward model with a system identification module is able to control these systems (Cheetah video: link-C.I.Ch. All videos are in SM Table A.2).

Model-based reinforcement learning. In our second approach to model-based control, we jointly trained a GN model and a policy function using SVG (Heess et al., 2015), where the model was used to backpropagate error gradients to the policy in order to optimize its parameters. Crucially, our SVG agent does not use a pre-trained model, but rather

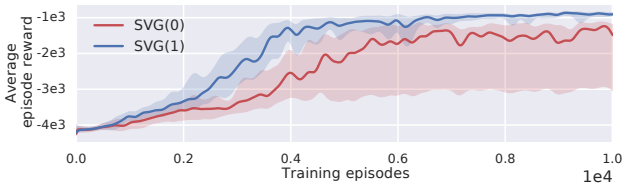

Figure 12. Learning curves for Swimmer6 SVG agents. The GN-based agent (blue) asymptotes earlier, and at a higher performance, than the model-free agent (red). The lines represent median performance for 6 random seeds, with 25 and 75% confidence intervals.

the model and policy were trained simultaneously.[8] Compared to a model-free agent (SVG(0)), our GN-based SVG agent (SVG(1)) achieved a higher level performance after fewer episodes (Figure 12). For GN-based agents with more than one forward step (SVG(2-4)), however, the performance was not significantly better, and in some cases was worse (SVG(5+)).

8. Discussion

This work introduced a new class of learnable forward and inference models, based on "graph networks" (GN), which implement an object- and relation-centric inductive bias. Across a range of experiments we found that these models are surprisingly accurate, robust, and generalizable when used for prediction, system identification, and planning in challenging, physical systems.

While our GN-based models were most effective in systems with common structure among bodies and joints (e.g., Swimmers), they were less successful when there was not much opportunity for sharing (e.g., Cheetah). Our approach also does not address a common problem for model-based planners that errors compound over long trajectory predictions.

Some key future directions include using our approach for control in real-world settings, supporting simulation-to-real transfer via pre-training models in simulation, extending our models to handle stochastic environments, and performing system identification over the structure of the system as well as the parameters. Our approach may also be useful within imagination-based planning frameworks (Hamrick et al., 2017; Pascanu et al., 2017), as well as integrated architectures with GN-like policies (Wang et al., 2018).

This work takes a key step towards realizing the promise of model-based methods by exploiting compositional representations within a powerful statistical learning framework, and opens new paths for robust, efficient, and general-purpose patterns of reasoning and decision-making.

[8] In preliminary experiments, we found little benefit of pre-training the model, though further exploration is warranted.

References

Amos, B., Dinh, L., Cabi, S., Rothrl, T., Muldal, A., Erez, T., Tassa, Y., de Freitas, N., and Denil, M. Learning awareness models. *ICLR*, 2018.

Atkeson, C. G. and Santamaria, J. C. A comparison of direct and model-based reinforcement learning. In *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, volume 4, pp. 3557–3564. IEEE, 1997.

Battaglia, P., Pascanu, R., Lai, M., Rezende, D. J., et al. Interaction networks for learning about objects, relations and physics. In *NIPS*, pp. 4502–4510, 2016.

Battaglia, P. W., Hamrick, J. B., and Tenenbaum, J. B. Simulation as an engine of physical scene understanding. *Proceedings of the National Academy of Sciences*, 110 (45):18327–18332, 2013.

Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4): 18–42, 2017.

Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

Chang, M. B., Ullman, T., Torralba, A., and Tenenbaum, J. B. A compositional object-based approach to learning physical dynamics. *arXiv preprint arXiv:1612.00341*, 2016.

Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

Craik, K. J. W. *The nature of explanation*, volume 445. CUP Archive, 1967.

Dai, H., Dai, B., and Song, L. Discriminative embeddings of latent variable models for structured data. In *ICML*, pp. 2702–2711, 2016.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In *NIPS*, pp. 3844–3852, 2016.

Deisenroth, M. and Rasmussen, C. Pilco: A model-based and data-efficient approach to policy search. In *ICML 28*, pp. 465–472. Omnipress, 2011.

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. Convolutional networks on graphs for learning molecular fingerprints. In *NIPS*, pp. 2224–2232, 2015.

Ehrhardt, S., Monszpart, A., Mitra, N. J., and Vedaldi, A. Learning a physical long-term predictor. *arXiv preprint arXiv:1703.00247*, 2017.

Fragkiadaki, K., Agrawal, P., Levine, S., and Malik, J. Learning visual predictive models of physics for playing billiards. *CoRR*, abs/1511.07404, 2015.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *ICML*, pp. 1263–1272, 2017.

Grzeszczuk, R., Terzopoulos, D., and Hinton, G. Neuroanimator: Fast neural network emulation and control of physics-based models. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pp. 9–20. ACM, 1998.

Gu, S., Lillicrap, T. P., Sutskever, I., and Levine, S. Continuous deep q-learning with model-based acceleration. *CoRR*, abs/1603.00748, 2016.

Hamrick, J. B., Ballard, A. J., Pascanu, R., Vinyals, O., Heess, N., and Battaglia, P. W. Metacontrol for adaptive imagination-based optimization. *arXiv preprint arXiv:1705.02670*, 2017.

Heess, N., Wayne, G., Silver, D., Lillicrap, T., Erez, T., and Tassa, Y. Learning continuous control policies by stochastic value gradients. In *NIPS*, pp. 2944–2952, 2015.

Henaff, M., Bruna, J., and LeCun, Y. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

Houthooft, R., Chen, X., Duan, Y., Schulman, J., Turck, F. D., and Abbeel, P. Curiosity-driven exploration in deep reinforcement learning via bayesian neural networks. *CoRR*, abs/1605.09674, 2016.

Johnson-Laird, P. N. Mental models in cognitive science. *Cognitive science*, 4(1):71–115, 1980.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. In *ICLR*, 2014.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Levine, S. and Abbeel, P. Learning neural network policies with guided policy search under unknown dynamics. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q. (eds.), *NIPS 27*, pp. 1071–1079. Curran Associates, Inc., 2014.

Li, W. and Todorov, E. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *ICINCO (1)*, pp. 222–229, 2004.

Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. In *ICLR*, 2016.

Miall, R. C. and Wolpert, D. M. Forward models for physiological motor control. *Neural networks*, 9(8):1265–1279, 1996.

Nagabandi, A., Kahn, G., Fearing, R. S., and Levine, S. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. *arXiv preprint arXiv:1708.02596*, 2017.

Niepert, M., Ahmed, M., and Kutzkov, K. Learning convolutional neural networks for graphs. In *ICML*, pp. 2014–2023, 2016.

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

Peng, X. B., Andrychowicz, M., Zaremba, W., and Abbeel, P. Sim-to-real transfer of robotic control with dynamics randomization. *CoRR*, abs/1710.06537, 2017.

Rajeswaran, A., Ghotra, S., Levine, S., and Ravindran, B. Epopt: Learning robust neural network policies using model ensembles. *CoRR*, abs/1610.01283, 2016.

Raposo, D., Santoro, A., Barrett, D., Pascanu, R., Lillicrap, T., and Battaglia, P. Discovering objects and their relations from entangled scene representations. *arXiv preprint arXiv:1702.05068*, 2017.

Rezende, D. J., Mohamed, S., and Wierstra, D. Stochastic backpropagation and approximate inference in deep generative models. In *ICML 31*, 2014.

Santoro, A., Raposo, D., Barrett, D. G., Malinowski, M., Pascanu, R., Battaglia, P., and Lillicrap, T. A simple neural network module for relational reasoning. In *NIPS*, pp. 4974–4983, 2017.

Scarselli, F., Yong, S. L., Gori, M., Hagenbuchner, M., Tsoi, A. C., and Maggini, M. Graph neural networks for ranking web pages. In *Web Intelligence, 2005. Proceedings. The 2005 IEEE/WIC/ACM International Conference on*, pp. 666–672. IEEE, 2005.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. Computational capabilities of graph neural networks. *IEEE Transactions on Neural Networks*, 20 (1):81–102, 2009a.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009b.

Schmidhuber, J. Curious model-building control systems. In *Proc. Int. J. Conf. Neural Networks*, pp. 1458–1463. IEEE Press, 1991.

Spelke, E. S. and Kinzler, K. D. Core knowledge. *Developmental science*, 10(1):89–96, 2007.

Sun, Y., Gomez, F. J., and Schmidhuber, J. Planning to be surprised: Optimal bayesian exploration in dynamic environments. In *AGI*, volume 6830 of *Lecture Notes in Computer Science*, pp. 41–51. Springer, 2011.

Tassa, Y., Erez, T., and Smart, W. D. Receding horizon differential dynamic programming. In *NIPS*, pp. 1465–1472, 2008.

Tassa, Y., Mansard, N., and Todorov, E. Control-limited differential dynamic programming. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 1168–1175. IEEE, 2014.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. d. L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 5026–5033. IEEE, 2012.

Wang, T., Liao, R., Ba, J., and Fidler, S. Nervenet: Learning structured policy with graph neural networks. *ICLR*, 2018.

Watters, N., Zoran, D., Weber, T., Battaglia, P., Pascanu, R., and Tacchetti, A. Visual interaction networks: Learning a physics simulator from video. In *NIPS*, pp. 4542–4550, 2017.

Yu, W., Liu, C. K., and Turk, G. Preparing for the unknown: Learning a universal policy with online system identification. *CoRR*, abs/1702.02453, 2017.