Learning to Control Self-Assembling Morphologies: A Study of Generalization via Modularity

Deepak Pathak [*1] **Chris Lu** [*1] **Trevor Darrell** [1] **Phillip Isola** [2] **Alexei A. Efros** [1]

Abstract

Contemporary sensorimotor learning approaches typically start with an existing complex agent (e.g., a robotic arm), which they learn to control. In contrast, this paper investigates a modular co-evolution strategy: a collection of primitive agents learns to dynamically self-assemble into composite bodies while also learning to coordinate their behavior to control these bodies. Each primitive agent consists of a limb with a motor attached at one end. Limbs may choose to link up to form collectives. When a limb initiates a link-up action and there is another limb nearby, the latter is magnetically connected to the 'parent' limb's motor. This forms a new single agent, which may further link with other agents. In this way, complex morphologies can emerge, controlled by a policy whose architecture is in explicit correspondence with the morphology. We evaluate the performance of these *dynamic* and *modular* agents in simulated environments. We demonstrate better generalization to test-time changes both in the environment, as well as in the agent morphology, compared to static and monolithic baselines. Project videos and code are available at `https://pathak22.github.io/modular-assemblies/`.

1. Introduction

Only a tiny fraction of the Earth's biomass is composed of higher-level organisms capable of complex sensorimotor actions of the kind popular in contemporary robotics research (navigation, pick and place, etc). A large portion are primitive single-celled organisms (Bar-On et al., 2018), such as bacteria. Possibly the single most pivotal event in the history of evolution was the point when single-celled organisms switched from always competing with each other for resources to sometimes cooperating, first by forming colonies, and later by merging into multicellular organisms (Alberts et al., 1994). These modular self-assemblies were successful because they combined the high adaptability of single-celled organisms while making it possible for vastly more complex behaviours to emerge. Indeed, one could argue that it is this modular design which allowed the multicellular organisms to successfully adapt, increase in complexity, and generalize to the constantly changing environment of prehistoric Earth. Like many researchers before us (Sims, 1994; Tu & Terzopoulos, 1994; Yim et al., 2000; Murata & Kurokawa, 2007; Yim et al., 2007), we are inspired by the biology of multicellular evolution as a model for emergent complexity in artificial agents. Unlike most previous work however, we are primarily focused on modularity as a way of improving adaptability and generalization to *novel test-time scenarios*.

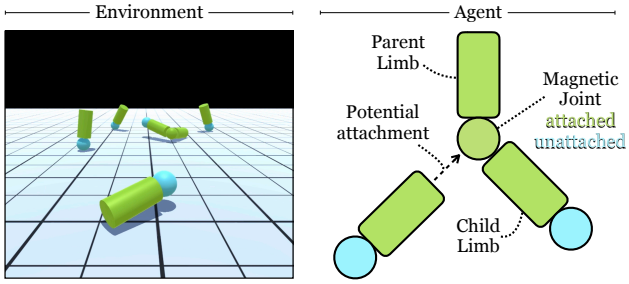


Figure 1. This work investigates the joint learning of control and morphology in self-assembling agents. Several primitive agents, containing a cylindrical body with a configurable motor, are dropped in a simulated environment (left). These primitive agents can self-assemble into collectives using magnetic joints (right).

In this paper, we present a study of modular self-assemblies of primitive agents — "limbs" — which can link up to solve a task. Limbs have the option to bind together by a magnet that connects their morphologies within magnetic range (Figure 1), and when they do so, they pass messages and share rewards. Each limb comes with a simple neural net that controls the torque applied to its joints. Linking and unlinking are treated as dynamic actions so that the limb assembly can change shape within an episode. Similar setup has previously been explored in robotics as "self-reconfiguring modular robots" (Stoy et al., 2010). However, unlike prior work on such robots, where the control policies are hand-defined, we show how to *learn* the policies and study the generalization properties that emerge.

[*]Equal contribution [1]UC Berkeley [2]MIT. Correspondence to: Deepak Pathak <pathak@cs.berkeley.edu>.

Preprint. Under review.

Our self-assembled agent can be represented as a graph of primitive limbs. Limbs pass messages to their neighbors in this graph in order to coordinate behavior. All limbs share a common policy function, parametrized by a neural network, which takes the messages from adjacent limbs as input and outputs a torque to rotate the limb in addition to the linking/un-linking action. We call the aggregate neural network a Dynamic Graph Network (DGN) since it is a graph neural network (Scarselli et al., 2009) that can dynamically change topology as a function of its own outputs.

We test our dynamic limb assemblies on two separate tasks: standing up and locomotion. We are particularly interested in assessing how well can the assemblies generalize to novel testing conditions, not seen at training, compared to static and monolithic baselines. We evaluate test-time changes to both the environment (changing terrain geometry, environmental conditions), as well as the agent itself (changing the number of available limbs). We show that the dynamic self-assembles are better able to generalize to these changes than the baselines. For example, we find that a single modular policy is able to control multiple possible morphologies, even those not seen during training, e.g., a 6-limb policy, trained to build a 6-limb tower, can be applied at test time on 3 or 12 limbs, and still able to perform the task.

The main contributions of this paper are:

- Training primitive agents that self-assemble into complex morphologies to jointly solve control tasks.
- Formulating morphological search as a reinforcement learning problem, where linking and unlinking are treated as actions.
- Representing policy via a graph whose topology matches the agent's physical structure.
- Demonstrating that self-assembling agents both train and generalize better than fixed-morphology baselines.

2. Environment and Agents

Investigating the co-evolution of control (i.e., *software*) and morphology (i.e., *hardware*) is not supported within standard benchmark environments typically used for sensorimotor control, requiring us to create our own. We opted for a minimalist design for our agents, the environment, and the reward structure, which is crucial to ensuring that the emergence of limb assemblies with complex morphologies is not forced, but happens naturally.

Environment Structure Our environment contains an arena where a collection of primitive agent limbs can self-assemble to perform control tasks. This arena is a ground surface equipped with gravity and friction. The arena can be procedurally changed to generate a variety of novel terrains by changing the height of each tile on the ground (see Figure 2). To evaluate the generalization properties of our agents, we generate a series of novels terrains. This include generating bumpy terrain by randomizing the height of nearby tiles, stairs by incrementally increasing the height of each row of tiles, hurdles by changing the height of each row of tiles, gaps by removing alternating rows of tiles, etc. Some variations also include putting the arena 'under water' which basically amounts to increased drag (i.e. buoyancy). During training, we start our environment with a set of six primitive limbs on the ground which can assemble to form collectives to perform complex tasks.

Agent Structure All primitive limbs share the same simple structure: a cylindrical body with a configurable motor on one end and the other end is free. The free-end of the limb can link up with the motor-end of the other limb, and then the motor acts as a joint between two limbs with three degrees of rotation. Hence, one can refer to the motor-end of the cylindrical limb as a *parent-end* and the free end as a *child-end*. Multiple limbs can attach their child-end to the parent-end of another limb, as shown in Figure 1, to allow for complex graph morphologies to emerge. The limb of the parent-end controls the torques of joint. The unlinking action can be easily implemented by detaching two limbs, but the linking action has to deal with the ambiguity of which limb to connect to (if at all). To resolve these modeling issues, we implement the linking action by attaching the closest limb within a small radius around the parent-node. The attachment mechanism is driven by a magnet inside the parent node which forces the closest child-limb within the magnetic range node to get docked onto itself if the parent signals to connect. If no other limb is present within the magnetic range, the linking action has no effect.

The primitive limbs are dropped in an environment to jointly solve a given control task. One key component of the self-assembling agent setup that makes it different from typical multi-agent scenarios (Wooldridge, 2009) is that if some agents assemble to form a collective, the resulting morphology becomes a new *single agent* and all limbs within the morphology maximize a joint reward function. The output action space of each primitive agent contains the continuous torque values that are to be applied to the motor connected to the agent, and are denoted by $\{\tau_\alpha, \tau_\beta, \tau_\gamma\}$ for three degrees of rotation. In addition to the torque controls, each limb can decide to attach another link at its parent-end, or decide to unlink its child-end if already connected to other limb. The linking and unlinking decisions are binary. This complementary role assignment of child and parent ends, i.e., parent can only link and child can only unlink, makes it possible to decentralize the control across limbs.

Sensory Inputs In our self-assembling setup, each agent limb only has access to its local sensory information and

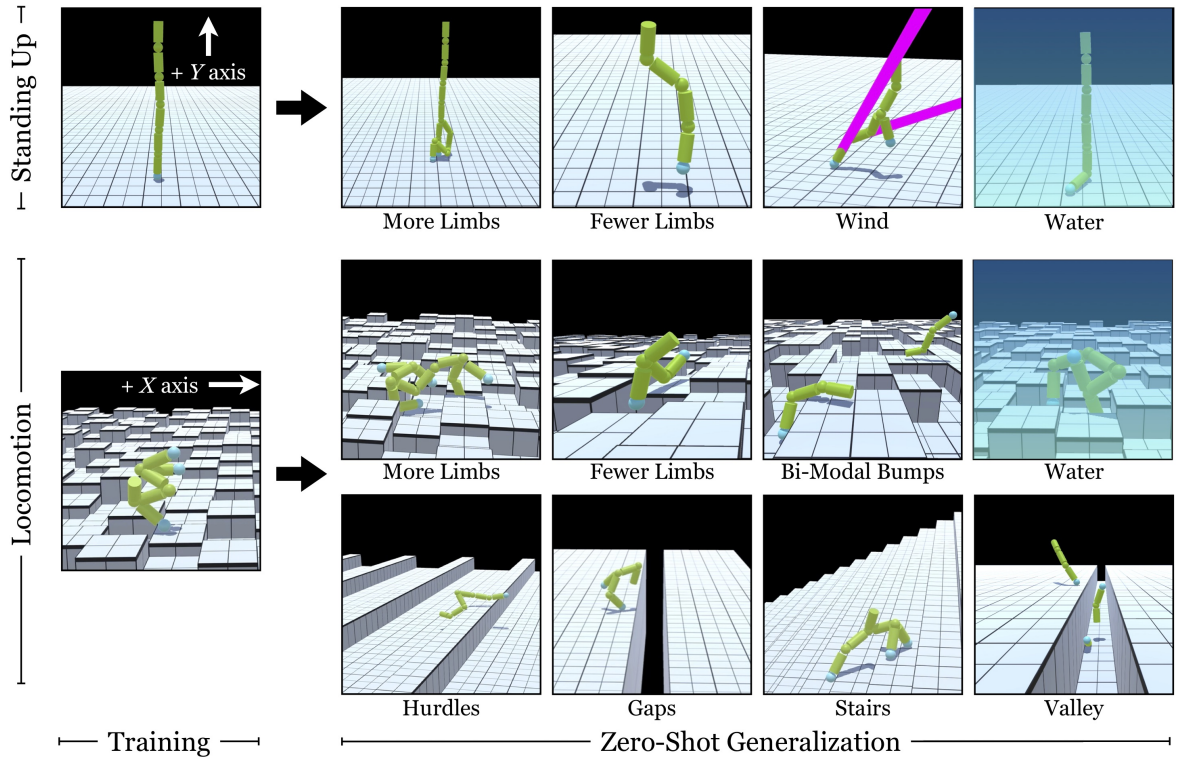


Figure 2. We illustrate our dynamic agents in two environments / tasks: standing up and locomotion. For each of these, we generate several new environment for evaluating generalization. Project videos at https://pathak22.github.io/modular-assemblies/.

does not know about other limbs. The sensory input of each agent includes its own dynamics, i.e., the location of the limb in 3-D euclidean coordinates, its velocity, angular rotation and angular velocity. Each end of the limb also has a trinary touch sensor to detect whether the end of the cylinder is touching 1) the floor, 2) another limb, or 3) nothing. Additionally, we also provide our limbs with a point depth sensor that captures the surface height on a 9×9 grid around the projection of center of limb on the surface.

One essential requirement to operationalize this setup is an efficient simulator to allow simultaneous simulation of several of these primitive limbs. We implement our environments in the Unity ML (Juliani et al., 2018) framework, which is one of the dominant platforms for designing realistic games. For computational reasons, we do not allow the emergence of cycles in the self-assembling agents by not allowing the limbs to link up with already attached limbs within the same morphology. However, our setup is trivially extensible to general graphs.

3. Learning to Control Self-Assemblies

Consider a set of primitive limbs indexed by i in $\{1, 2, \ldots, n\}$, which are dropped in the environment arena $\mathcal{E}$ to perform a given continuous control task. If needed, these limbs can assemble to form complex collectives in order to improve their performance on the task. The task is represented by a reward function r_t and the goal of the limbs is to maximize the discounted sum of rewards over time t. If some limbs assemble to form a collective, the resulting morphology effectively becomes a single agent with a combined network to maximize the combined reward of the connected limbs. Further, the reward of an assembled morphology is a function of the whole morphology and not the individual agent limbs. For instance, in the task of learning to stand up, the reward is the height of the individual limbs if they are separate, but is the height of the whole morphology if those limbs have assembled into a collective.

3.1. Co-evolution: Linking/Unlinking as an Action

To learn a modular controller policy that could generalize to novel setups, our agents must learn the controller jointly as the morphology evolves over time. The limbs should simultaneously decide which torques to apply to their respective motors, while taking into account the connected morphology. Our hypothesis is that if a controller policy could learn in a modular fashion over iterations of increasingly sophisticated morphologies (see Figure 4), it could learn to be robust and generalizable to diverse situations. So, how can we optimize control and morphology under a common end-to-end framework?

We propose to treat the decision of linking and unlinking as additional actions of our primitive limbs. The to-

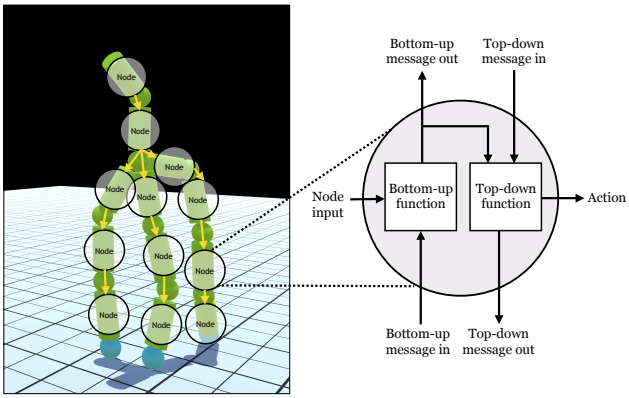


Figure 3. High-level visualization of our method. A set of primitive 'limbs' learn to self-assemble into morphologies where each limb is represented by a neural network linked via graph of physical edges. The inset on right shows the message-passing diagram for each node. Project videos at https://pathak22.github.io/modular-assemblies/.

tal action space a_t at each iteration t can be denoted as $\{\tau_\alpha, \tau_\beta, \tau_\gamma, \sigma_{link}, \sigma_{unlink}\}$ where τ_* denote the raw *continuous* torque values to be applied at the motor and σ_* denote the *binary* actions whether to connect another limb at the parent-end or disconnect the child-end from the other already attached limb. This simple view of morphological evolution allows us to use ideas from reinforcement learning (Sutton & Barto, 1998).

3.2. Modularity: Self-Assembly as a Graph of Limbs

Integration of control and morphology in a common framework is only the first step. The key question is how to model this controller policy such that it is modular and reuses information across generations of morphologies. Let a_t^i be the action space and s_t^i be the local sensory input-space of the agent i. One naive approach to maximizing the reward is to simply combine the states of the limbs into the input-space, and output all the actions jointly using a single network. Formally, the policy is simply $\vec{a}_t = [a_t^0, a_t^1 \dots a_t^n] = \Pi(s_t^0, s_t^0 \dots, s_t^n)$. This interprets the self-assemblies as a single monolithic agent, ignoring the graphical structure. This is the current approach to solve many control problems, e.g., Mujoco environments like humanoid (Brockman et al., 2016) where the policy Π is trained to maximize the sum of discounted rewards using reinforcement learning.

In this work, we represent the policy of the agent via a graph neural network (Scarselli et al., 2009) in such a way that it explicitly corresponds to the morphology of the agent. Let's consider the collection of primitive limbs as graph G where each node is a limb i. Two limbs being physically connected by a joint is analogous to having an edge in the graph. As mentioned above, each limb has two endpoints, a *parent-end* and a *child-end*. At a joint, the limb which connects via its parent-end acts as a parent-node in the corresponding edge, and the other limbs, which connect via their child-ends, are child-nodes. The parent-node (i.e., the agent with the parent-end) controls the torque of the edge (i.e., the joint motor), as described in Section-2.

3.3. Dynamic Graph Networks (DGN)

Each primitive limb node i has a policy controller of its own, which is represented by a neural network π_θ^i and receives a corresponding reward r_t^i for each time step t. We represent the policy of the self-assembled agent by the aggregated neural network that is connected in the same graphical manner as the physical morphology. The edge connectivity of the graph is represented in the overall graph policy by passing messages that flow from each limb to the other limbs physically connected to it via a joint. The parameters θ are shared across each primitive limbs allowing the overall policy of the graph to be modular with respect to each node. However, recall that the agent morphologies are dynamic, i.e., the connectivity of the limbs changes based on policy outputs. This changes the edge connectivity of the corresponding graph network at every timestep, depending on the actions chosen by each limb's policy network in the previous timestep. Hence, we call this aggregate neural net a *Dynamic Graph Network (DGN)* since it is a graph neural network that can dynamically change topology as a function of its own outputs in the previous iteration.

DGN Optimization A typical rollout of our self-assembling agents during an episode of training contains a sequence of torques τ_t^i and the linking actions σ_t^i for each limb at each timestep t. The policy parameters θ are optimized to jointly maximize the reward for each limb:

$$\max_\theta \sum_{i=\{1,2\dots,n\}} \mathbb{E}_{\vec{a}^i \sim \pi_\theta^i}[\Sigma_t r_t^i] \tag{1}$$

We optimize this objective via policy gradients, in particular, PPO (Schulman et al., 2017).

DGN Connectivity The topology is captured in the DGN by passing messages through the edges between individual network nodes. These messages allow each node to take into account its context relative to other nodes, and are supposed to convey information about the neighbouring policy network in the graph. Since the parameters of these limb networks are shared across each node, these messages can be seen as context information that may inform the policy of its role in the corresponding connected component of graph. The aggregated flow through the whole graph can be encapsulated by passing these contextual messages in topological order (no cycles). One can either do a top-down pass, beginning from the root node (i.e., the node with no parents) to the leaf nodes, or do bottom-up pass, from leaves

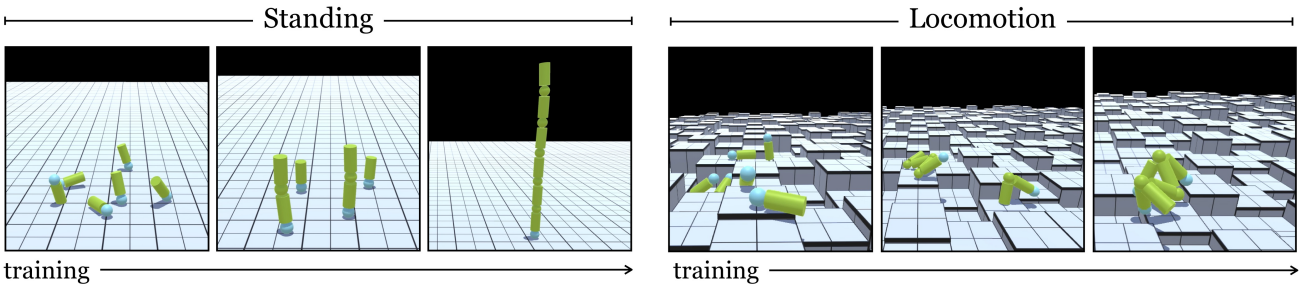


Figure 4. Co-evolution of Morphology w/ Control during Training: The gradual co-evolution of controller as well as the morphology of self-assembling agents over the course of training for the task of Standing Up (left) and Locomotion (right).

to root node. This idea is inspired from classical work on Bayesian graph networks where message passing is used for belief-propagation (Jordan, 2003). However, when the graph contains cycles, this idea can be easily extended by performing message-passing iteratively through the cycle until convergence, similar to loopy-belief-propagation in Bayesian graphs (Murphy et al., 1999). We now discuss these message-passing strategies:

(a) Top-down message passing: Instead of defining π^i_θ to be just as a function of state, $\pi^i_\theta : s^i_t \rightarrow a^i_t$, we pass each limb's policy network information about its parent node as well, denoted by the message $m^{p_i}_t$, where p_i is the parent of node i. This limb in turn outputs a message m^i_t to its own children. Formally, we define $\pi^i_\theta : [s^i_t, m^{p_i}_t] \rightarrow [a^i_t, m^i_t]$. If i has no parents (i.e, root), a vector of zeros is passed in $m^{p_i}_t$. This is computed recursively until the messages reach the leaf nodes.

(b) Bottom-up message passing: In this strategy, messages are passed from leaf nodes to root, i.e., each agent gets information from its children, but not from its parent. Similar to top-down, we redefine π^i_θ as $\pi^i_\theta : [s^i_t, m^{C_i}_t] \rightarrow [a^i_t, m^i_t]$ where m^i_t is the output message of policy that goes into the parent limb and $m^{C_i}_t$ is the aggregated input messages from all the children nodes, i.e, $m^{C_i}_t = \sum_{c \in C_i} m^c_t$. If i has no children (i.e, root), a vector of zeros is passed in $m^{C_i}_t$. Messages are passed recursively until the root node.

(c) Bottom-up then top-down message passing: In this strategy, we pass messages both ways: bottom-up, then top-down. In the absence of cycles in graph, a one-way pass (either top-down or bottom-up) is sufficient to capture the aggregated information, similar to Bayesian trees (Jordan, 2003). Even though both-way message-passing is redundant, we still explore it as an alternative since it might help in learning when the agent grows too complex. This is implemented by dividing the policy into two parts, each responsible for one direction of message passing, i.e., the parameters $\theta = [\theta_1, \theta_2]$. First the bottom-up message passing is formulated as $\pi^i_{\theta_1} : [s^i_t, m^{C_i}_t] \rightarrow m^i_t$ where the sensory input s^i_t and input messages $m^{C_i}_t$ are used to generate outgoing messages to the parent node. In the top-down pass, messages from the parent are used, in addition with the agent's own message, to output its action: $\pi^i_{\theta_2} : [m^i_t, m^{p_i}_t] \rightarrow [a^i_t, \hat{m}^i_t]$ where $\hat{m}^i_t$ are the messages passed to the children nodes.

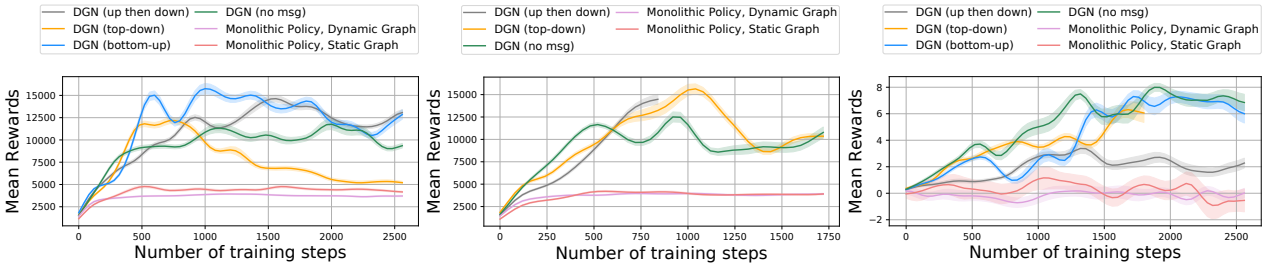
(d) No message passing: Note that for some environments or tasks, the context from the other nodes might not be a necessary requirement for effective control. In such scenarios, message passing might creates extra overhead for training a DGN. Importantly, even with no messages being passed, the DGN framework still allows for coordination between limbs. This is similar to a typical cooperative multi-agent setup (Wooldridge, 2009) where each limb makes its own decisions in response to the previous actions of the other agents. However, our setup differs in that our agents may physically join up, rather than just coordinating behavior. To implement the no message passing variant of DGN, we simply zero-out the messages $m^{p_i}_t, m^i_t$ at each timestep t.

Pseudo-code for the DGN algorithm is in the appendix.

4. Implementation Details and Baselines

Implementation Details: We use PPO (Schulman et al., 2017) as the underlying reinforcement learning method to optimize Equation 1. Each limb policy is represented by 4-layered fully-connected neural network and trained with a learning rate of $3e-4$, discount factor of 0.995, entropy coefficient of 0.01 and batch size of 2048. Parameters are shared across network modules and they all predict action at the same time. Each episode is 5000 steps long at training. Across all the tasks, the number of limbs at training is kept fixed to 6. Limbs start each episode disconnected and located just above the ground plane at random locations, as shown in Figure 4. In the absence of an edge, input messages are set to 0 and the output ones are ignored. Action space is continuous raw torque values. We take the model from each time step and evaluate it on 50 episodes to plot mean and standard-deviation in training curves. At test, we report the mean reward across 50 episodes of 1200 environment steps.

Baselines We compare the role of the above four message passing strategies in DGN across a variety of tasks. Dif-



(a) Standing Up (b) Standing Up w/ Wind (c) Locomotion

Figure 5. Training self-assembling agents: We show the performance of different methods for joint training of control and morphology for three tasks: learning to stand up (left), standing up in the presence of wind (center) and locomotion in bumpy terrain (right). These policies generalize to novel scenarios as shown in respective tables.

ferent strategies may work well in different scenarios. We further compare how well these dynamic morphologies perform in comparison to a learned monolithic policy for both dynamic and fixed morphologies. In particular, we compare to a (a) *Monolithic Policy, Dynamic Graph*: in this baseline, our agents are still dynamic and self-assemble to perform the task, however, their controller is represented by a single monolithic policy that takes as input the combined state of all agents and outputs actions for each of them. (b) *Monolithic Policy, Fixed Graph*: For each task, a hand-designed morphology is constructed from the limbs and trained using a single monolithic policy that takes as input the combined state of all agents and outputs the actions for all agents. The agents are not able to combine or separate This can be compared to a standard robotics setup in which a morphology is predefined and then a policy is learned to control it. Note that one cannot generalize *Monolithic Policy* baselines to scenarios where the number of limbs vary as it would change the action and state space of the policy.

For the *Fixed Graph* baseline, we chose the fixed morphology to be a straight line chain of 6-limbs (i.e., a linear morphology) in all the experiments including the task of standing up and locomotion. This linear-chain may be optimal for standing as tall as possible, but it is not necessarily optimal for learning to stand; the same would hold for locomotion. Further, note that, the best performing DGN variants also converges to linear-chain morphology (shown in Figure 4 and video results on the project website) to achieve the best reward in case of standing up task. Moreover, one can confirm that the locomotion task is also solvable with linear-morphology because one of the DGN ablation methods converged to a linear-morphology while doing well at locomotion (see video).

5. Experiments

We test the co-evolution of morphology and control across two primary tasks where self-assembling agents learn to: (a) stand up, and (b) perform locomotion. We first investigate if modular co-evolution results in the emergence of complex self-assemblies. But the main focus of our investigation is to evaluate if the emerged modular controller generalizes to novel scenarios. We study test-time generalization to 1) novel agent morphologies, and 2) novel environments.

5.1. Learning to Self-Assemble

We first validate if it is possible to train self-assembling agent policy end-to-end via Dynamic Graph Networks. In this section, we discuss each of our environments and compare the training efficiency of each method.

Standing Up Task In this task, each agent's objective is to maximize the height of the highest point in its morphology. Limbs have an incentive to self-assemble because the potential reward would scale with the number of limbs if the self-assembled agent can control them. The training process begins with six-limbs falling on the ground randomly, as shown in Figure 4. These limbs act independently in the beginning but gradually learn to self-assemble as training proceeds. Figure 5a compares the training efficiency and performance of different methods during training. We found that our DGN policy variants perform significantly better than the monolithic policies for the standing up task. In particular, the bottom-up and up-then-down message passing strategies attain the highest reward. To further understand the poor performance of the monolithic agent, we ran an ablation study, training a monolithic policy by varying the number of limbs. We found that the monolithic policy works well with up to 4-limb morphology but fails to learn 6-limb agents, as shown in Appendix Figure 6.

Standing Up in the Wind Task Same as the previous task, except with the addition of 'wind', which we operationalize as random forces applied to random points of each limb at random times, see Figure 2(Wind). The training curves in Figure 5b show the superior performance of DGN with respect to other baselines.

Locomotion Task The reward function for locomotion is defined as the distance covered by the agent along X-axis. The training is performed on a bumpy terrain shown in Figure 2. The training performance in Figure 5c shows

Environment	DGN (up then down)	DGN (top-down)	DGN (bottom-up)	DGN (no msgs)	Monolithic Policy (dynamic graph)	Monolithic Policy (fixed graph)
			Standing Up			
Training Environment						
Standing Up	15253	13486	**17518**	12470	4104	5351
Zero-Shot Generalization						
More (2x) Limbs	15006 (98%)	14429 (107%)	**19796 (113%)**	14084 (113%)	–	–
Fewer (.5x) Limbs	**11730 (77%)**	9842 (73%)	10839 (62%)	9070 (73%)	–	–
			Standing Up in the Wind			
Training Environment						
Standing Up in Wind	16339	**18423**	–	17237	4176	4500
Zero-Shot Generalization						
2x Limbs + (S)Winds	**16250 (99%)**	15351 (83%)	–	15728 (91%)	–	–
			Locomotion			
Training Environment						
Locomotion	3.91	6.87	8.71	**9.0**	0.96	2.96
Zero-Shot Generalization						
More (2x) Limbs	4.01 **(103%)**	4.29 (63%)	5.47 (63%)	**9.19** (102%)	–	–
Fewer (.5x) Limbs	3.52 (90%)	4.49 (65%)	6.64 (76%)	**8.2 (91%)**	–	–

Table 1. Zero-Shot Generalization to Number of Limbs: We show quantitative evaluation of the generalization ability of the learned policies. For every method, we first pick the best performing model from the training and then evaluate it on each of the novel scenarios without any further finetuning, i.e., in a zero-shot manner. We report the score attained by the self-assembling agent along with the percentage of training performance retained upon transfer in parenthesis. Higher value is better.

that DGN variants outperform the monolithic baselines (see appendix for ablation). Interestingly, DGN variant with no message passing performs the best. Upon investigation, we found that it is possible to do well on this locomotion task with a large variety of morphologies, unlike the task of standing up where a tower morphology is optimal. Any morphology with sufficient height and forward velocity is able to make competitive progress in locomotion (see videos), and thus reducing message-passing to an unnecessary overhead. As discussed in Section 3.3, no message passing merely implies the absence of context to the limbs, but the DGN aggregated policy is still modular and jointly learned with the morphology over the episode.

5.2. Zero-Shot Generalization to Number of Limbs

We investigate if our trained policy generalizes to changes the number of limbs used in the assembly. We pick the best model from training and evaluate its performance without anyz finetuning at test-time, i.e., zero-shot generalization.

Standing Up Task We train the policy with 6 limbs and test with 12 and 4 limbs. Total reward is shown in Table 1. Despite different number of limbs than training, all the DGN variants are able to retain similar performance. The co-evolution of morphology jointly with the controller allows the modular policy to experience increasingly complex morphological structures. We hypothesize that such a natural morphology curriculum at training makes the agent more robust at test-time. Note that *monolithic policy* baselines cannot be generalized to more or fewer limbs due to the fixed action and state space. We also refer the reader to project videos on the website.

Standing Up in the Wind Task Similarly, we evaluate the agent policy trained for standing up task in winds with 6 limbs to 12 limbs. Results in Table 1 show that the DGN both-ways messaging passing variant is the most robust. It could be that, in the presence of push-n-pulls, both-ways communication is more helpful because a random force on a limb can affect all other attached limbs.

Locomotion Task We also evaluate the generalization of locomotion policies trained with 6 limbs to 12 and 4 limbs. As shown in Table 1, DGN variant no-message passing not only achieved the best performance at training but is also able to retain most of its performance following similar reasoning as discussed in the previous subsection.

5.3. Zero-Shot Generalization to Novel Environments

In addition to morphology, we also evaluate the performance of our modular agents in novel environments. We create several different scenarios by varying environment conditions (described in Section 2) to test zero-shot generalization.

Standing Up Task We test our trained policy without any further finetuning in environments with increased drag (i.e., 'under water'), and adding varying strength of random push-n-pulls (i.e. , 'wind'). Table 2 shows that DGN seems to generalize better than monolithic policies. We believe that this generalization is result of both the morphology being dynamic as well as the fact that they learned to assemble in physical conditions (e.g. forces like gravity) with gradually growing morphologies. Such forces with changing morphology are similar to setup with varying forces acting on fixed morphology resulting in robustness to external interventions like winds.

Environment	DGN (up then down)	DGN (top-down)	DGN (bottom-up)	DGN (no msgs)	Monolithic Policy (dynamic graph)	Monolithic Policy (fixed graph)
			Standing Up			
Training Environment						
Standing Up	15253	13486	**17518**	12470	4104	5351
Zero-Shot Generalization						
Water + 2x Limbs	16642 **(109%)**	14192 (105%)	**16871** (96%)	13360 (107%)	–	–
Winds	14654 (96%)	12116 (90%)	**16803** (96%)	12560 **(101%)**	3923 (96%)	4531 (85%)
Strong Winds	14727 (97%)	13416 **(99%)**	**15853** (90%)	12257 (98%)	3937 (96%)	4961 (93%)
			Standing Up in the Wind			
Training Environment						
Standing Up in Wind	16339	**18423**	–	17237	4176	4500
Zero-Shot Generalization						
(S)trong Winds	15649 (96%)	**17384** (94%)	–	–	4010 (96%)	4507 **(100%)**
Water + 2x(L) + (S)Winds	**17254 (106%)**	17068 (93%)	–	16592 (96%)	–	–
			Locomotion			
Training Environment						
Locomotion	3.91	6.87	8.71	**9.0**	0.96	2.96
Zero-Shot Generalization						
Water + 2x Limbs	2.64 (68%)	3.54 (52%)	6.57 (75%)	**7.2 (80%)**	–	–
Hurdles	1.84 (47%)	3.66 (53%)	**6.39 (73%)**	5.56 (62%)	-0.77 (-79%)	-3.12 (-104%)
Gaps in Terrain	1.84 (47%)	2.8 (41%)	3.25 (37%)	**4.17** (46%)	-0.32 (-33%)	2.09 **(71%)**
Bi-modal Bumps	2.97 (76%)	4.55 (66%)	**6.62 (76%)**	6.15 (68%)	-0.56 (-57%)	-0.44 (-14%)
Stairs	1.0 (26%)	4.25 (62%)	6.6 (76%)	**8.59 (95%)**	-8.8 (-912%)	-3.65 (-122%)
Inside Valley	4.37 **(112%)**	**6.55** (95%)	5.29 (61%)	6.21 (69%)	0.47 (48%)	-1.35 (-45%)

Table 2. Zero-Shot Generalization to Novel Environments: The best performing model from the training is evaluated on each of the novel scenarios without any further finetuning. The score attained by the self-assembling agent is reported along with the percentage of training performance retained upon transfer in parenthesis. Higher value is better.

Standing Up in the Wind Task Similarly, the policies trained with winds are able to generalize to scenarios with either stronger winds or winds inside water. Interestingly, all message-passing schemes generalize differently.

Locomotion Task We generate several novel scenarios for evaluating locomotion: with water, a terrain with hurdles of a certain height, a terrain with gaps between platforms, a bumpy terrain with a bi-modal distribution of bump heights, stairs, and an environment with a valley surrounded by walls on both sides (see Figure 2). These variations are generated procedurally as discussed in Section 2. Across these novel environments, the modular policies learned by DGN tend to generalize better than the monolithic agent policies, as indicated in Table 2.

This generalization could be explained by the incrementally increasing complexity of self-assembling agents at training. For instance, the training begins with all limbs separate which gradually form group of two, three and so on, until the training converges. Since the policy is modular, the training of smaller size assemblies with small bumps would in turn prepare the large assemblies for performing locomotion through higher hurdles, stairs etc at test. Furthermore, the training terrain has a finite length which makes the self-assemblies launch themselves forward as far as possible upon reaching the boundary to maximize the distance along X-axis. This behavior helps the limbs generalize to environments like gaps or valley where they end up on the next terrain upon jumping and continue to perform locomotion.

6. Related Work

Morphologenesis & self-reconfiguring modular robots The idea of modular and self-assembling agents goes back at least to Von Neumman's *Theory of Self-Reproducing Automata* (Von Neumann et al., 1966). In robotics, such systems have been termed "self-reconfiguring modular robots" (Stoy et al., 2010; Murata & Kurokawa, 2007). There has been a lot of work in modular robotics to design real hardware robotic modules that can be docked together to form complex robotic morphologies (Yim et al., 2000; Wright et al., 2007; Romanishin et al., 2013; Gilpin et al., 2008; Daudelin et al., 2018). We approach this problem from a learning perspective, in particular deep RL, and study the resulting generalization properties.

A variety of alternative approaches have also been proposed to optimize agent morphologies, including genetic algorithms that search over a generative grammar (Sims, 1994), as well as directly optimizing controllers by minizing energy-based objectives (De Lasa et al., 2010; Wampler & Popović, 2009). A learning-based alternative is to condition policy on several hardwares to ensure robustness (Chen et al., 2018). One key difference between these approaches and our own is that we achieve morphogenesis via *dynamic actions* (linking), which agents take during their lifetimes, whereas the past approaches treat morphology as an optimization target to be updated between generations or episodes. Since the physical morphology also defines the connectivity of the policy net, our proposed algorithm can

also be viewed as performing a kind of neural architecture search (Zoph & Le, 2016) in physical agents.

Graph neural networks Encoding graphical structures into neural networks has been used for a large number of applications, including visual question answering (Andreas et al., 2016), quantum chemistry (Gilmer et al., 2017), semi-supervised classification (Kipf & Welling, 2016), and representation learning (Yang et al., 2018). The works most similar to ours involve learning control policies (Wang et al., 2018; Sanchez-Gonzalez et al., 2018). For example, Nervenet (Wang et al., 2018) represents individual limbs and joints as nodes in a graph and demonstrates multi-limb generalization, just like our system does. However, the morphologies on which Nervenet operates are not learned jointly with the policy and hand-defined to be compositional in nature. Others (Battaglia et al., 2018; Huang et al., 2018) have shown that graph neural networks can also be applied to inference models as well as to planning. Many of these past works implement some variant of Graph Neural Networks (Scarselli et al., 2009) which operate on general graphs. Our method leverages the constraint that the morphologies can always be represented as a rooted tree in order to simplify the message passing.

Concurrent Work Ha (2018); Schaff et al. (2018) use reinforcement learning to improve limb design given fixed morphology. Alternatively, Wang et al. (2019) gradually evolves the environment to improve robustness of an agent. However, both the work assume the topology of agent morphology to stay the same during train and test.

7. Discussion

Modeling intelligent agents as modular, self-assembling morphologies has long been a very appealing idea. The efforts to create practical systems to evolve artificial agents goes back at least two decades to the beautiful work of Karl Sims (Sims, 1994). In this paper, we are revisiting these ideas using the contemporary machinery of deep networks and reinforcement learning. Examining the problem in the context of machine learning, rather than optimization, we are particularly interested in modularity as a key to generalization, in terms of improving adaptability and robustness to novel environmental conditions. Poor generalization is the Achilles heel of modern robotics research, and the hope is that this could be a promising direction in addressing this key issue. We demonstrated a number of promising experimental results, suggesting that modularity does indeed improve generalization in simulated agents. While these are just the initial steps, we believe that the proposed research direction is promising and its exploration will be fruitful to the research community. To encourage follow-up work, we will publicly release all code, models, and environments.

Acknowledgments

We would like to thank Igor Mordatch, Chris Atkeson, Abhinav Gupta and the members of BAIR for fruitful discussions and comments. This work was supported in part by Berkeley DeepDrive, and the Valrhona reinforcement learning fellowship. DP is supported by Facebook graduate fellowship.

References

Alberts, B., Bray, D., Lewis, J., Raff, M., Roberts, K., and Watson, J. D. *Molecular Biology of the Cell*. Garland Publishing, New York, 1994. 1

Andreas, J., Rohrbach, M., Darrell, T., and Klein, D. Neural module networks. In *CVPR*, 2016. 9

Bar-On, Y. M., Phillips, R., and Milo, R. The biomass distribution on earth. *Proceedings of the National Academy of Sciences*, 2018. 1

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018. 9

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. *arXiv:1606.01540*, 2016. 4

Chen, T., Murali, A., and Gupta, A. Hardware conditioned policies for multi-robot transfer learning. In *NIPS*, 2018. 8

Daudelin, J., Jing, G., Tosun, T., Yim, M., Kress-Gazit, H., and Campbell, M. An integrated system for perception-driven autonomy with modular robots. *Science Robotics*, 2018. 8

De Lasa, M., Mordatch, I., and Hertzmann, A. Feature-based locomotion controllers. In *ACM Transactions on Graphics (TOG)*, 2010. 8

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*, 2017. 9

Gilpin, K., Kotay, K., Rus, D., and Vasilescu, I. Miche: Modular shape formation by self-disassembly. *IJRR*, 2008. 8

Ha, D. Reinforcement learning for improving agent design. *arXiv preprint arXiv:1810.03779*, 2018. 9

Huang, D.-A., Nair, S., Xu, D., Zhu, Y., Garg, A., Fei-Fei, L., Savarese, S., and Niebles, J. C. Neural task graphs: Generalizing to unseen tasks from a single video demonstration. *arXiv preprint arXiv:1807.03480*, 2018. 9

Jordan, M. I. An introduction to probabilistic graphical models, 2003. 5

Juliani, A., Berges, V.-P., Vckay, E., Gao, Y., Henry, H., Mattar, M., and Lange, D. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*, 2018. 3

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 9

Murata, S. and Kurokawa, H. Self-reconfigurable robots. *IEEE Robotics & Automation Magazine*, 2007. 1, 8

Murphy, K. P., Weiss, Y., and Jordan, M. I. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, 1999. 5

Romanishin, J. W., Gilpin, K., and Rus, D. M-blocks: Momentum-driven, magnetic modular robots. In *IROS*, 2013. 8

Sanchez-Gonzalez, A., Heess, N., Springenberg, J. T., Merel, J., Riedmiller, M., Hadsell, R., and Battaglia, P. Graph networks as learnable physics engines for inference and control. *arXiv preprint arXiv:1806.01242*, 2018. 9

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. The graph neural network model. *IEEE Transactions on Neural Network*, 2009. 2, 4, 9

Schaff, C., Yunis, D., Chakrabarti, A., and Walter, M. R. Jointly learning to construct and control agents using deep reinforcement learning. *arXiv preprint arXiv:1801.01432*, 2018. 9

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017. 4, 5

Sims, K. Evolving virtual creatures. In *Computer graphics and interactive techniques*, 1994. 1, 8, 9

Stoy, K., Brandt, D., Christensen, D. J., and Brandt, D. *Self-reconfigurable robots: an introduction*. Mit Press Cambridge, 2010. 1, 8

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press Cambridge, 1998. 4

Tu, X. and Terzopoulos, D. Artificial fishes: Physics, locomotion, perception, behavior. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, 1994. 1

Von Neumann, J., Burks, A. W., et al. Theory of self-reproducing automata. *IEEE Transactions on Neural Networks*, 1966. 8

Wampler, K. and Popović, Z. Optimal gait and form for animal locomotion. In *ACM Transactions on Graphics (TOG)*, 2009. 8

Wang, R., Lehman, J., Clune, J., and Stanley, K. O. Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions. *arXiv preprint arXiv:1901.01753*, 2019. 9

Wang, T., Liao, R., Ba, J., and Fidler, S. Nervenet: Learning structured policy with graph neural networks. *ICLR*, 2018. 9

Wooldridge, M. *An introduction to multiagent systems*. John Wiley & Sons, 2009. 2, 5

Wright, C., Johnson, A., Peck, A., McCord, Z., Naaktgeboren, A., Gianfortoni, P., Gonzalez-Rivero, M., Hatton, R., and Choset, H. Design of a modular snake robot. In *IROS*, 2007. 8

Yang, Z., Dhingra, B., He, K., Cohen, W. W., Salakhutdinov, R., LeCun, Y., et al. Glomo: Unsupervisedly learned relational graphs as transferable representations. *arXiv preprint arXiv:1806.05662*, 2018. 9

Yim, M., Duff, D. G., and Roufas, K. D. Polybot: a modular reconfigurable robot. In *ICRA*, 2000. 1, 8

Yim, M., Shen, W.-M., Salemi, B., Rus, D., Moll, M., Lipson, H., Klavins, E., and Chirikjian, G. S. Modular self-reconfigurable robot systems. *IEEE Robotics & Automation Magazine*, 2007. 1

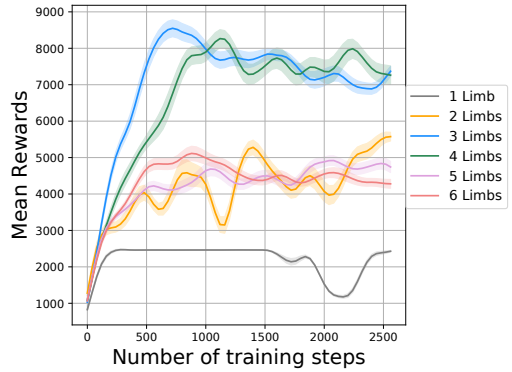
Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016. 9

A. Appendix

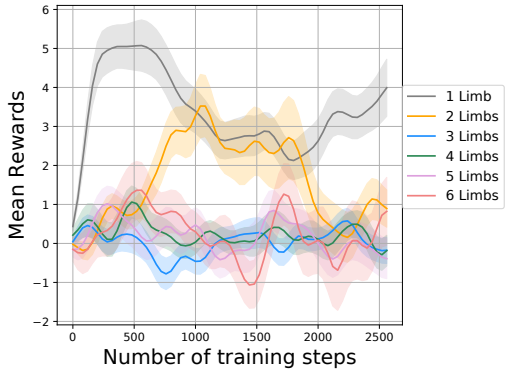
In this section, we provide additional details about the experimental setup and extra preliminary experiments.

A.1. Performance of Fixed-Graph Baseline vs. Number of Limbs

To verify whether the training of *Monolithic Policy w/ Fixed Graph* is working, we ran it on standing up and locomotion tasks across varying number of limbs. We show in Figure 6 that the baseline performs well with less number of limbs which suggests that the reason for failure in 6-limbs case is indeed the morphology graph being fixed, and not the implementation of this baseline.



(a) Standing Up



(b) Locomotion

Figure 6. The performance of *Monolithic Policy w/ Fixed Graph* baseline as the number of limbs varies in the two tasks: standing up (left) and locomotion (right). This shows that the monolithic baseline works well with less (1-3 limbs), but fails with 6 limbs during training.

A.2. Pseudo Code of the DGN Algorithm

Algorithm 1 Notation Summary (defined in Section 3.3)

foreach *node* i **do**

$\pi^i_{\theta_1}(s^i_t, m^{C_i}_t) = m^i_t$

$\pi^i_{\theta_2}(m^i_t, m^{p_i}_t) = [a^i_t, \hat{m}^i_t]$

end

where

s^i_t: observation state of agent limb i

a^i_t: action output of agent limb i: 3 torques, attach, detach

$m^{C_i}_t$: aggregated message from children nodes input to agent i (bottom-up-1)

m^i_t: output message that agent i passes to its parent (bottom-up-2)

$m^{p_i}_t$: message from parent node to agent i (top-down-1)

$\hat{m}^i_t$: message from agent i to its children

θ: θ_1, θ_2

messages are 32 length floating point vectors.

Algorithm 2 Pseudo-code: Bottom-up, Top-down DGN

Initialize parameters θ_1, θ_2 randomly.
Initialize all message vectors $m_t^{C_i}, m_t^i, m_t^{p_i}, \hat{m}_t^i$ to be zero
Represent graph connectivity G as a list of edges
Note: In the beginning, all edges are zeros, i.e., non-existent
foreach *timestep* t **do**
  Each limb agent i observes its own state vector s_t^i
  foreach *agent* i **do**
    # Compute incoming child messages
    $m_t^{C_i} = 0$
    foreach *child node* c *of agent* i **do**
      $m_t^{C_i} += m_t^c$
    end
    # Compute message to parent p of agent i in G
    $m_t^i := \pi_{\theta_1}^i(s_t^i, m_t^{C_i})$
    # Compute action and messages to children of agent i
    $a_t^i, \hat{m}_t^i := \pi_{\theta_2}^i(m_t^i, m_t^{p_i})$
    # Execute morphology change as per a_t^i
    if $a_t^i[3] ==$ *attach* **then**
      find closest agent j within distance d from agent i, otherwise j=NULL
      add edge (i, j) in G
      also make physical joint between (i, j)
    end
    if $a_t^i[4] == detach$ **then**
      delete edge (i, parent of i) in G
      also delete physical joint between (i, j)
    end
    # Execute torques from a_t^i
    Apply torques $a_t^i[0], a_t^i[1], a_t^i[2]$
  end
  # Update message variables
  foreach *agent* i **do**
    let p be parent of agent i in G
    if p *is NULL* **then**
      $m_t^{p_i} = 0$
    else
      $m_t^{p_i} = \hat{m}_t^i$
    end
  end
  # Update graph and agent morphology
  Find all connected components in G
  foreach *connected component* c **do**
    foreach *agent* $i \in c$ **do**
      reward r_t^i = reward of c (e.g. max height)
    end
  end
end
Update $\theta = \theta_1, \theta_2$ to maximize discounted reward using PPO as follows:
let $\vec{a}_t = [a_t^1, a_t^2..a_t^n]$
$\vec{s}_t = [s_t^1, s_t^2..s_t^n]$
$\hat{A}_t =$ advantage of discounted rewards, $r_t = \sum_{agenti} r_t^i$
PPO: $\max_\theta \mathbb{E}[\hat{A}_t \frac{\pi_\theta(\vec{a}_t|\vec{s}_t)}{\pi_{\theta_{\text{old}}}(\vec{a}_t|\vec{s}_t)} - \beta \text{KL}(\pi_{\theta_{\text{old}}}(.|\vec{s}_t)\pi_\theta(.|\vec{s}_t))]$
Repeat until training converges

Algorithm 3 Pseudo-code: Bottom-up DGN

Same as Algorithm-2 but hard-code incoming parent messages to be always 0, i.e., $m_t^{p_i} = 0$ in each iteration.

Algorithm 4 Pseudo-code: Top-down DGN

Same as Algorithm-2 but hard-code incoming child messages to be always 0, i.e., $m_t^{C_i} = 0$ in each iteration.

Algorithm 5 Pseudo-code: No-message DGN

Same as Algorithm-2 but hard-code incoming child and parent messages to be always 0, i.e., $m_t^{C_i} = 0$ and $m_t^{p_i} = 0$ in each iteration.