LEARNING TO CONTROL SELF-ASSEMBLING MORPHOLOGIES: A STUDY OF GENERALIZATION VIA MODULARITY

Anonymous authors
Paper under double-blind review

ABSTRACT

Much of contemporary sensorimotor learning assumes that one is already given a complex agent (e.g., a robotic arm) and the goal is to learn to control it. In contrast, this paper investigates a modular co-evolution strategy: a collection of primitive agents learns to self-assemble into increasingly complex collectives in order to jointly solve control tasks. Each primitive agent consists of a limb and a neural controller. When two limbs link up, a joint is added between them, and messages are passed between the two neural controllers. Linking is treated as a dynamic action: to solve a task, agents learn how to self-assemble into a joint morphology and a joint neural net. The resulting policies and morphologies are *dynamic* and *modular*, which allows them to better generalize to novel test-time environments. Experiments in several simulated environments are presented; see project videos at https://doubleblindICLR19.github.io/self-assembly/.

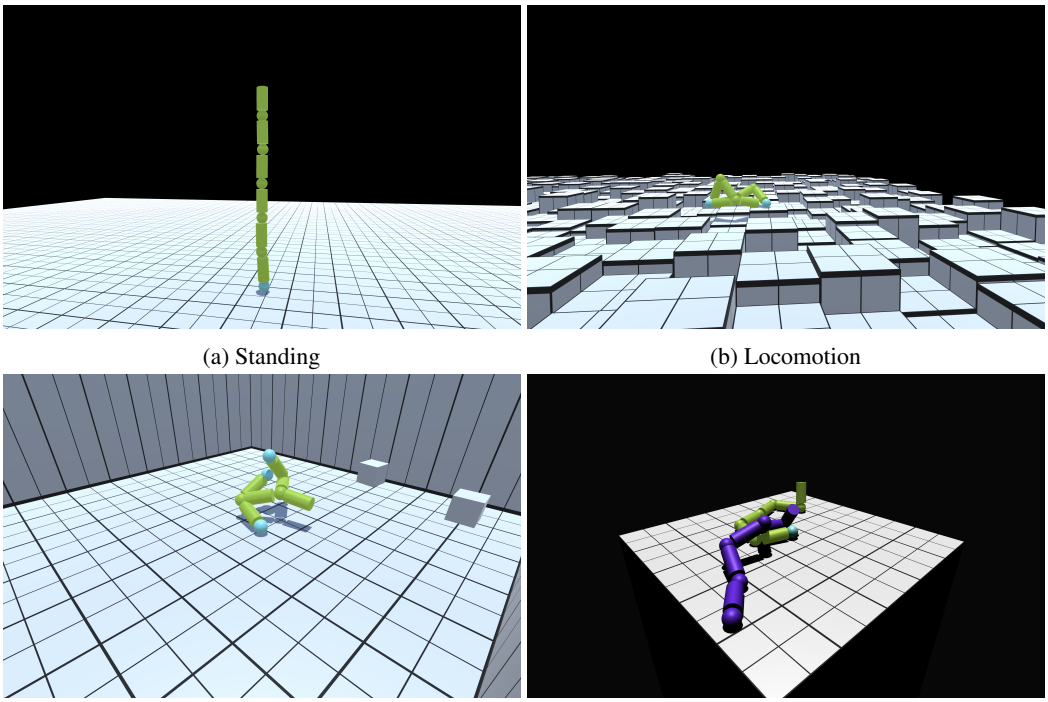
1 INTRODUCTION

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 is primitive single-celled organisms, such as bacteria (Bar-On et al., 2018). 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. Like many researchers before us (Murata & Kurokawa, 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 evaluating modularity in the context of learning agents, as a way of improving adaptability and robustness to novel environmental conditions.

In this paper, we present a study of modular self-assemblies of primitive agents – "limbs" which can link up to solve a shared task. The limbs have the option to bind together by adding a joint that connects their morphologies, and when they do so, they share rewards. Linking and unlinking is treated as a dynamic action, so that the limb assembly can change shape during a single episode of the simulation. This results in what is known as a "self-reconfiguring modular robot" (Stoy et al., 2010). Our contribution is to demonstrate how to optimize such a system using deep reinforcement learning, and to study the generalization properties of the policies and morphologies that emerge.

To make this problem tractable, we restrict the limb assemblies to be rooted trees. 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 as input messages from adjacent limbs and outputs a torque to rotate the limb. 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 demonstrate that the aggregate multi-limb agent exhibits *modularity* and *compositionality*, in the sense that the component limbs can be rearranged while retaining a high degree of functionality. The



(a) Standing (b) Locomotion

(c) Manipulation (d) Sumo

Figure 1: Visualization of different environments / tasks: (a) stand, (b) locomotion, (c) manipulation (pushing), and (d) self-play (Sumo). See project videos at https://doubleblindICLR19.github.io/self-assembly/.

upshot is that the trained system is robust to perturbations in the morphology, and exhibits improved generalization on novel tasks, compared to fixed morphology baselines.

This work studies one way primitive agents may learn (or evolve) to bind together into larger collectives. The algorithm performs a physical architecture search, finding dynamic morphologies that are effective at solving tasks. Since the physical morphology also defines the connectivity of the policy net, the algorithm can also be viewed as performing a kind of neural architecture search. Our experiments suggest that this self-assembling neuro-physical architecture has advantages for generalization to new environments and tasks.

2 Environment

We propose a multiagent reinforcement learning environment in which individual agents combine to form larger collectives to solve control tasks. The environments, agents, and tasks are structured as follows.

Agent Structure Each agent is given a state of its environment based off of the features used in (Heess et al., 2017). Also included in the state is the relative location of the nearest agent it can join with. Each agent also has five continuous actions. Three of them are for the joint torques the agent applies to itself. The fourth action allows the agent to combine with the nearest agent, as long as the nearest agent is within a specified range and the fifth action detaches the agent from its parent if it is connected to one.

Task descriptions We present four different tasks: Locomotion, Standing, Manipulation, and Sumo. In the Locomotion task, each agent's reward is proportional to its velocity on the x-axis. This encourages each agent to make forward progress; however, it also encourages the agents to combine to overcome obstacles when needed. In the Standing task, each agent's reward is proportional the highest vertical point in its morphology. In this task, the agents have an incentive to combine since

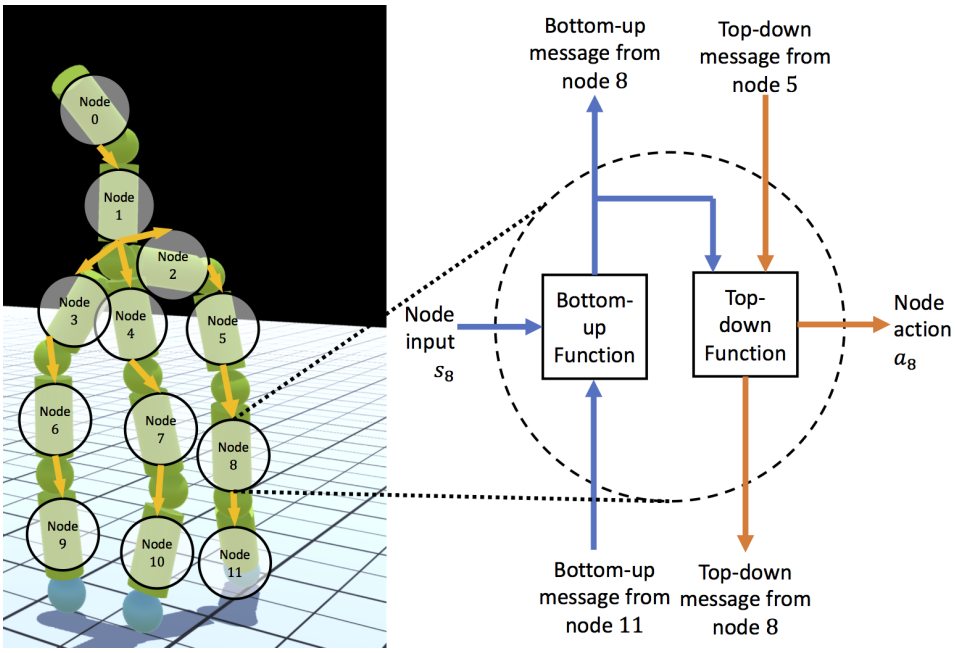


Figure 2: High-level visualization of our method. A set of primitive 'limbs' learn to self-assemble into morphologies, with each limb being a node in a Directed Acyclic Graph (DAG). The inset on right shows the message-passing diagram for each node. Project videos at https://doubleblindICLR19.github.io/self-assembly/.

the potential reward scales with the number of agents in the body. In the Manipulation task, two blocks are placed in the environment. Each agent's reward is negatively proportional to the distance between the blocks. This is to encourage the behavior of pushing the blocks together. Finally, the Sumo task is a self-play game based on (Bansal et al., 2017), in which agents are grouped into two teams and receive reward upon the opposing team's limbs falling out of the arena.

Environments We train for the Locomotion task on a simple environment in which segments of the floor have uniformly sampled heights from a specified range. We will show that from just training on this environment, the method generalizes to very different environments. This includes a stairs-like environment, environments with gaps in the floor, an environment with large hurdles, and more. Many environments for Locomotion were inspired by (Heess et al., 2017). We also show that the Standing and Manipulation tasks also generalize to variations of the simple initial environment. These variations include simulated wind, as well as increased drag to simulate movement in a dense fluid. Furthermore, for each of these tasks we demonstrate that they can operate, at test time, with an arbitrary number of limbs. Finally, for the sumo task, we construct a simple square arena similar to (Bansal et al., 2017).

3 Dynamic Graph Networks (DGN)

We represent a collection of agents by a modular graph (more specifically, a rooted tree) in which each node represents a single agent and each edge represents a physical joint between them. For each node i, we consider the standard reinforcement learning setting in which an agent with a parametrized policy π^i_θ takes in an observation s^i_t at time t and outputs an action $a^i_t = \pi^i_\theta(s^i_t)$. The environment then calculates a following state $s^i_{t+1} = T(s^i_t, a^i_t)$ as well as a reward $r^i_t = R(s^i_t, a^i_t)$. The objective is to maximize the discounted sum of rewards, $J(\theta) = E(\sum_{t=0}^{\infty} \gamma^t r_t)$. To do this, we employ Proximal Policy Optimization (PPO), by (Schulman et al., 2017). PPO has several advantages over many policy gradient methods. Standard policy gradient methods attempt to estimate the gradient steps of a policy in order to directly optimize the expected sum of rewards using stochastic gradient descent – unfortunately, these methods are usually sensitive to step size and can suffer heavily from high variance in the gradients. PPO attempts to alleviate this problem by optimizing a surrogate objective

function: $L = \mathbb{E}[\min(l_t(\theta)A_t, clip(l_t(\theta), 1-\epsilon, 1+\epsilon)A_t))]$ where $l_t(\theta)$ is the likelihood ratio $l_t(\theta) = \frac{\pi_\theta(a_t|s_t}{\pi_{\theta_{old}}(a_t|s_t)}$ and A_t is the generalized advantage estimate. Dynamic Graph Networks (DGN) encode a dynamic structure over the policy to take advantage of the modularity and compositionality inherent in the environment.

The naive approach to maximizing the objective in the environment is to simply combine all of the states and actions and interpret them as a single monolithic agent, ignoring the graphical structure. This is the current approach to many control problems for environments such as Mujoco's humanoid. More formally, the policy is simply $\sum_i a_t^i = \pi_\theta(\sum_i s_t^i)$. Another naive approach is to give each agent i an independent policy $\pi_\theta^i(s_a^i)$ for each node. While this is modular, it assumes that there is no relationship between the agents, which is clearly false in the case of morphology. In order to encode the relationship between the agents, edges need to be added to the graph.

For each node to take into account its context relative to other nodes, it needs to receive information about its neighbouring nodes in the graph. Thus, rather than defining π^i to be $\pi^i : s^i \rightarrow a^i$, it should also take in information about its parent. More formally, we redefine π^i as $\pi^i : s^i, m^p \rightarrow a^i$ where p is the parent of node i. However, this also means that the node should pass information onto its children. So, we need to define m^i for each node i. We simply append this to the output of π^i. Thus, we finally define $\pi^i : s^i, m^p \rightarrow a^i, m^i$.

Given that the agent's morphology is in the form of a rooted tree, there are two ways in which one can pass messages between them: Bottom-up and top-down. In the bottom-up direction, each agent passes messages to its parent and then each node i then calculates a_t^i. Unfortunately, this means that leaf nodes do not receive any contextual information from their parents. This can be fixed with a top-down approach in which each agent passes a message to its child. Unfortunately, with this approach, the root does not receive contextual information to act on.

To solve this, we propose doing both bottom-up and then top-down. This way, each node gets complete contextual information–each node can obtain its location in the graph and receive information from any other node in the graph via some message path. We choose to do bottom-up and then top-down rather than top-down and then bottom-up since that method would not guarantee the ability for all nodes to receive information from each other node before deciding an action. We call the bottom-up direction "Sensory Transmission" and the top-down direction "Motor Transmission".

Sensory transmission is formulated as follows:

Let the sensory transmission module operation be f_m and its output be m_i for an agent i. For each agent i, $m_i = f_m(s_i, \sum_{c \in C_a} m_c)$ where s_i is the observation of agent i and C_i is the set of its children. If a less than the maximum number of children, a vector of zeros is passed in instead for each missing child. This is evaluated recursively in the tree until m_i is calculated for each agent i.

Motor transmission is similarly calculated:

Let the motor transmission operation be f_n and its output be a_i for an agent i where a_i is the actions as motor torques as well as the value estimation for the agent. For each agent i, $a_i = f_n(m_i, a_p)$ where p is the parent of a. If i has no parents, a vector of zeros is passed in instead. This is evaluated recursively until a_i is calculated for each agent i.

Note that f_s and f_m are not specific to each agent. Instead, the module shares its weights across all agents. This allows the method to generalize to different number of agents and different starting positions.

A potential downside of the bottom-up top-down approach to the problem is the information bottleneck that occurs in the root. One can observe that information from all networks gets condensed into the root. The idea is that the immediate local information matters more, and both m_a and n_p correspond to information that comes directly from its children and parents, respectively.

4 Baseline Methods

We compare the Dynamic Graph Network with a number of ablations and baseline methods to understand and observe its effectiveness.

PPO w/ Fixed Morphology: For each task, a hand-designed morphology was constructed from the limbs and trained with PPO using a single monolithic agent that takes in the state of all agents and outputs the actions for all agents. The agents are not able to combine or separate and no modular structures were used. This can be compared to a standard robotics task setup in which the morphology is predefined and the controls are learned for it.

PPO w/ Dynamic Morphology: This is similar to the above baseline except the morphology is dynamic. This means that each agent is capable of combining and separating; however, a single monolithic agent still takes in the state of all agents and outputs actions for them.

DGN w/o message passing: This is a simple variant in which case rather than having a single monolithic agent, each agent is treated separately. However, no message passing is done between the networks. This can be compared to a standard multi-agent setup where each limb acts in a modular manner.

DGN w/ top-down message passing: This is a variant of DGN in which we pass the messages top-down. This means that each agent node in the graph gets information from its parent, but not any information from its children. Since the sensory state space information is received from the children, this approach ignores the first part of processing performed by the children nodes.

DGN w/ bottom-up message passing: This is a varient of the DGN method in which a bottom-up message passing is done. This means that each agent gets information from its children, but not from its parent. Again, because the sensory state space information is received from the children, this approach accumulates the information from children and pass it to the respect parent node.

DGN w/ top-down, bottom-up message passing: This is an ablation of the DGN method in which only the bottom-up message passing is done. This means that each agent gets information from its children, but not from its parent. Note that passing messages both ways is redundant in case of a DAG (which is the case for our agents), but still keep it as an ablation as this might be required when the agent grows too complex.

5 Experimental Results

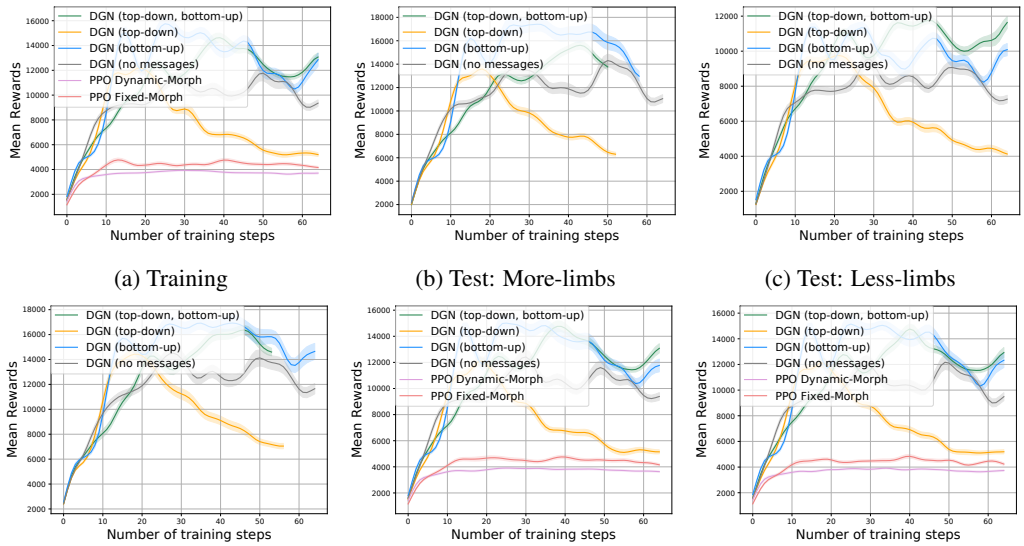
We have proposed a formulation for training agents with dynamic morphologies by self-assembly. We now test our hypothesis across a variety of 3 tasks: (a) agent learning to stand up; (b) agent learning to stand up in the presence of distractor objects, and (c) agent learning to perform locomotion in bumpy terrain.

The goal of results section is to two fold: first, to check whether our agent can learn to control self-assembling morphologies in a reinforcement learning setup where the agent is represented by a dynamic graph network. Second, to test if our learned morphologies generalize across environments and to unseen complications. Finally, we show that the method is capable of complex behavior through demonstrating its abilities in a sumo-like self play environment. We compare all the variants of our method across all the training environments as well as the generalization ones. Note that we cannot compare non-modular baselines in generalization environments that vary the number of limbs since the action and state spaces change. To test the generalization, we pick the trained model from different points in time and plot them across new environments without finetuning at all, in a zero-shot manner.

5.1 Standing task

The agent is tasked to stand up in this environment. The learning process begins by six-limbs falling on the ground randomly. At this point, they are independently controlled. Note that at train-time, we always use six limbs. This is so that we can test 'out-of-domain' generalization at test time by adding twice the number of limbs and see if the self-assembling process generalizes. The reward structure for the agent is equal to the height of the agent.

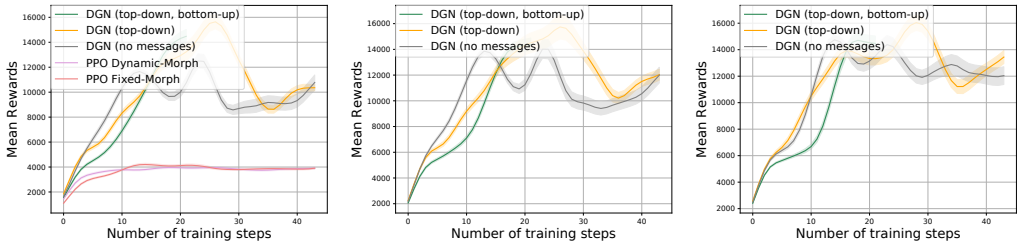
In this environment, we see that bottom-up DGN and two-pass DGN perform the best, whereas top-down DGN performs worse than no message passing. This is likely due to the fact that the



(a) Training (b) Test: More-limbs (c) Test: Less-limbs

(d) Test: Water w/ More-limbs (e) Test: Winds (f) Test: Stronger Winds

Figure 3: **Standing Task Generalization**: Performance of DGN (our method) across different environments.



(a) Test: Training (b) Test: (c) Test: Water + Winds + More

Figure 4: **Standing Task with Distractors Generalization**: Performance of DGN (our method) across different environments.

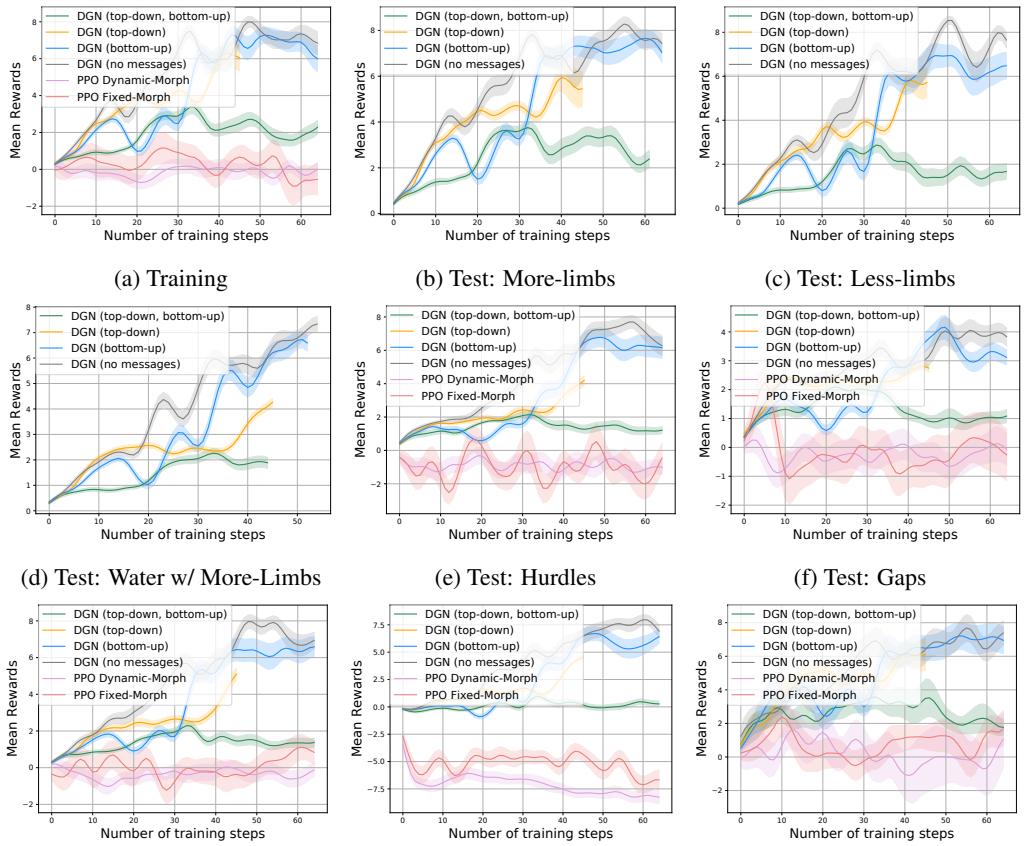
optimal policy is a simple tower of agents – message passing is only particularly useful in a bottom-up direction. However, the key thing to note that the obtained reward is almost the same in new environments as well which suggests generalization. A more clear picture of these results can be obtained by looking at the dynamically combining morphologies in the project video.

5.2 Standing Task in Presence of Distractors

In this environment, we trained on winds in order to obtain more robustness. We observe that the top-down method performs better in this environment compared to the Standing task without distractors. This is likely because in the presence of distractors, communication both ways can be helpful since a random force on a single limb affects all other agents.

5.3 Locomotion Task

In this environment, we observe that while modular methods outperform non-modular networks and fixed morphologies, message passing does not seem to help. This is likely due to the fact that agents do not combine as often in these environments and only do so to overcome specific obstacles. Thus, less message passing is needed and used overall. Furthermore, we also observe that modular methods successfully generalize to unseen environments.

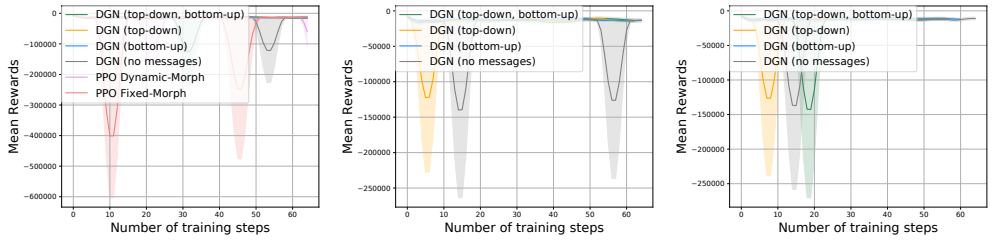


(a) Training (b) Test: More-limbs (c) Test: Less-limbs

(d) Test: Water w/ More-Limbs (e) Test: Hurdles (f) Test: Gaps

(g) Test: Bi-modal (h) Test: stairs (i) Test: Valley

Figure 5: **Locomotion Task Generalization**: Performance of DGN (our method) across different environments.



(a) Test: Training (b) Test: More-Limbs (c) Test: Water w/ More-Limbs

Figure 6: **Manipulation Task Generalization**: Performance of DGN (our method) across different environments.

5.4 MANIPULATION TASK

In this environment, we observe that bottom-up and two-pass DGN again perform the most consistently across the environments. This is likely due to the fact that small errors can result in large negative reward. As a result of this, contextual information is necessary in order to minimize any errors due to lack of state knowledge.

5.5 SELF-PLAY WRESTLING BETWEEN TWO TEAMS

Since the self-play was symmetrical in training, reward plots for each team becomes meaningless. Instead, we obtained more qualitative results that can be observed in the video. Overall, it appears as though dynamicism allows for more complex policies and that the DGN is able to achieve this.

6 RELATED WORK

Morphologenesis and 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). Our main contribution is to apply deep RL to this setting, 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 over morphology parameters with RL (Schaff et al., 2018). One key difference between these alternative approaches and our own is that we achieve morphogenesis via *dynamic actions* (linking), which agents take during their lifetimes, whereas the alternatives treat morphology as an optimization target to be updated between generations or episodes.

Graph neural networks Encoding graphical structures into neural networks has been used for a large number of applications, including 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 controls for systems. For example, Nervenet (Wang et al., 2018) represents individual limbs and joints as nodes in a graph as well and demonstrates multi-limb generalization. However, the morphologies on which it operates are 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 forward and inference models as well as planning. Many of these 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.

7 CONCLUSION

Modeling intelligent agents as modular, self-assembling morphologies has long been a very appealing idea, with efforts to create practical systems going 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 demonstrate 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 release all code, models, and environments online once the paper is published.

REFERENCES

Bruce Alberts, Dennis Bray, Julian Lewis, Martin Raff, Keith Roberts, and James D Watson. *Molecular Biology of the Cell*. Garland Publishing, New York, 1994. 1

Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. Emergent complexity via multi-agent competition. *CoRR*, 2017. 3

Yinon M Bar-On, Rob Phillips, and Ron Milo. The biomass distribution on earth. *Proceedings of the National Academy of Sciences*, pp. 201711842, 2018. 1

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018. 8

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*, 2017. 8

Nicolas Heess, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, Ali Eslami, Martin Riedmiller, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017. 2, 3

De-An Huang, Suraj Nair, Danfei Xu, Yuke Zhu, Animesh Garg, Li Fei-Fei, Silvio Savarese, and Juan Carlos Niebles. Neural task graphs: Generalizing to unseen tasks from a single video demonstration. *arXiv preprint arXiv:1807.03480*, 2018. 8

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 8

Satoshi Murata and Haruhsa Kurokawa. Self-reconfigurable robots. *IEEE Robotics & Automation Magazine*, 14(1):71–78, 2007. 1, 8

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Network*, 2009. 1, 8

Charles B. Schaff, David Yunis, Ayan Chakrabarti, and Matthew R. Walter. Jointly learning to construct and control agents using deep reinforcement learning. *CoRR*, abs/1801.01432, 2018. URL http://arxiv.org/abs/1801.01432. 8

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347. 3

Karl Sims. Evolving virtual creatures. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pp. 15–22. ACM, 1994. 8

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

John Von Neumann, Arthur W Burks, et al. Theory of self-reproducing automata. *IEEE Transactions on Neural Networks*, 5(1):3–14, 1966. 8

Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler. Nervenet: Learning structured policy with graph neural networks. 2018. 8

Zhilin Yang, Bhuwan Dhingra, Kaiming He, William W Cohen, Ruslan Salakhutdinov, Yann LeCun, et al. Glomo: Unsupervisedly learned relational graphs as transferable representations. *arXiv preprint arXiv:1806.05662*, 2018. 8