One Policy to Control Them All: Shared Modular Policies for Agent-Agnostic Control

Wenlong Huang [1] **Igor Mordatch** [2] **Deepak Pathak** [3 4]

A. Appendix

A.1. Result Videos

We show videos for all variants trained with a single policy on the project website: `https://huangwl18.github.io/modular-rl/`. We recommended referring to videos to observe how our single 4-layer network policy can represent different gate behaviors across different agent morphologies. One way message passing is sometimes able to learn for more than one morphology, but cannot represent multiple gates. However, both-way message passing is able to represent multiple gates due to *emergence of centralization from decentralized modules*.

A.2. Implementation and Training

We use TD3 (Fujimoto et al., 2018) as the underlying reinforcement learning method. The internal optimizer for TD3 is Adam (Kingma & Ba, 2015). The initial positions and velocities of each agent are randomized at the beginning of each episode. For the first $10,000$ time-steps during training, actions are uniformly sampled from the action space. The policy is trained with a learning rate of $4\mathrm{e}{-4}$, a tau of 0.046, and an exploration noise of 0.13. All internal modules are 4-layered fully-connected neural networks with ReLU and tanh non-linearity. The dimension of message vectors is 32. Messages are normalized before passed to the children and the parent.

For multi-morphology training, each morphology has its own environment and an independent replay buffer of size $1e6$. The maximum size of all the replay buffers is capped at $1e7$ and whenever there are $n > 10$ environments, each environment has a replay buffer of size of $1e7/n$. We run all environments in parallel using vectorized environment from OpenAI Baselines (Dhariwal et al., 2017). To speed up training and inference, we also use the dynamic batching package (Polosukhin & Zavershynskyi, 2018) when there is no dependency between modules, e.g. when the limbs are neither an ancestor nor descendent of each other.

[1]UC Berkelely [2]Google [3]CMU [4]Facebook AI Research. Correspondence to: Deepak Pathak <dpathak@cs.cmu.edu>.

Proceedings of the 37th International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020. Copyright 2020 by the author(s).

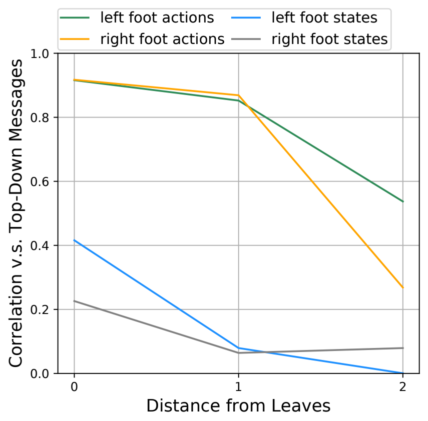

Figure 1. Message Range Analysis. We examine how far the messages reach and what relationship it has with the leaves' local states and actions by doing correlation analysis between leaves' local states/actions and the messages from its predecessors. The figure shows the closer a limb is to a leaf, the more its message contains the 'instructions' for the leaf.

For each single-category training (walker++, humanoid++, hopper++, and cheetah++), we use its default reward function from Gym. For multi-category training (walker-hopper++ and walker-hopper-humanoid++), we use the reward function from walker++ which consists of x-axis displacement (distance covered by agent), alive reward, and the sum of squared actions (for penalizing large action values).

A.3. Analysis of Message Propagation Range

In both-way message passing, messages are initially passed from the leaves to the root and then from the root back to the leaves. We here investigate whether the messages convey global information by showing the correlation between the states, actions, and the messages passed at different levels. Specifically, we test in the walker environment and we investigate how much the states and the actions are correlated with the furthest messages to the closest messages. Due to the different dimensionality, we first reduce the dimensions of the data to one by running Principle Component Analysis (PCA), and the final results are averaged over an episode.

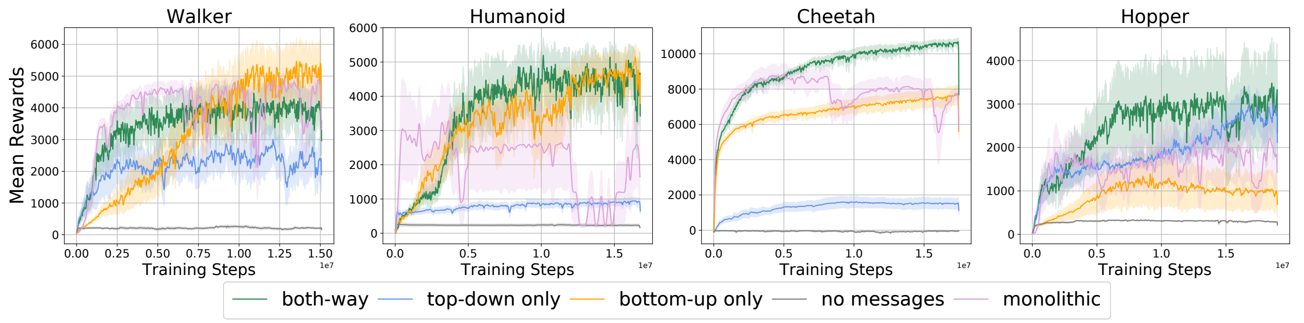

Figure 2. A sanity check that compares the average rewards on standard environments by the Shared Modular Policies with different message-passing schemes and the monolithic baseline. All the policies are only trained with one agent morphology at a time. The figure shows that the Shared Modular Policies with both-way message-passing can model multiple simple locomotion tasks just as well as a monolithic baseline.

Algorithm 1 Joint Training of All Agents

```
1:  **Notation Summary:**
2:  $NN_{bu}$: bottom-up module parameterized by $\theta_1$
3:  $NN_{td}$: top-down module parameterized by $\theta_2$
4:  $s_t^{(e)}$: all limbs' states of environment $e$ at time $t$
5:  $a_t^{(e)}$: all limbs' actions of environment $e$ at time $t$
6:  $r_t^{(e)}$: rewards of environment $e$ at time $t$
7:  $done^{(e)}$: whether environment $e$ is done
8:  $rb^{(e)}$: replay buffer for environment $e$
9:  **init:** SMP = ($NN_{bu}$, $NN_{td}$) from scratch.
10:       empty replay buffer $rb^{(e)}$ for each environment $e$.
11: **while** not converged **do**
12:     // collect one episode of data for all environments
13:     **for all** environment $e$ **do**
14:         **while** $e$ is not done **do**
15:             $a_t^{(e)} \leftarrow$ SMP($s_t^{(e)}$)
16:             $s_{t+1}^{(e)}, r_t^{(e)}, done^{(e)} \leftarrow$ simulate($e$, $a_t^{(e)}$)
17:             Add ($s_t^{(e)}, s_{t+1}^{(e)}, a_t^{(e)}, r_t^{(e)}, done^{(e)}$) to $rb^{(e)}$
18:         **end while**
19:     **end for**
20:     // train SMP for each environment one by one
21:     **for all** environment $e$ **do**
22:         SMP $\leftarrow$ trainWithTD3(SMP, $rb^{(e)}$)
23:     **end for**
24: **end while**
```

Algorithm 2 Both-way Shared Modular Policies (SMP)

```
1:  **Notation Summary:**
2:  $NN_{bu}$: bottom-up module parameterized by $\theta_1$
3:  $NN_{td}$: top-down module parameterized by $\theta_2$
4:  $s_i$: local states of limb $i$
5:  $a_i$: local action of limb $i$
6:  $m^{i\to p(i)}$: message passed from limb $i$ to the parent of $i$
7:  $m^{p(i)\to i}$: message passed from the parent of limb $i$ to $i$
8:  $\{m^{c\to i}\}_{c\in\mathcal{C}(i)}$: the set of all messages passed from the
    children of limb $i$ to $i$
9:  $\{m^{i\to c}\}_{c\in\mathcal{C}(i)}$: the set of different messages passed
    from limb $i$ to each of its children
10: **Input:**  environment $e$ and all limbs' states $\{s_i\}_{i=1}^k$
11: **Output:** actions for all limbs of that agent $\{a_i\}_{i=1}^k$
12: // get agent limbs in topological order (root to leaf)
13: nodeList $\leftarrow$ topologicalOrdering($e$)
14: // dynamically change the policy's graph structure to
    match that of the agent
15: SMP $\leftarrow$ changeGraph(SMP, nodeList)
16: // bottom-up message passing (leaf to root)
17: **for** node $i$ in reversed(nodeList) **do**
18:     **if** $i$ is leaf **then**
19:         $m^{i\to p(i)} \leftarrow NN_{bu}(s_i, \vec{0})$
20:     **else**
21:         $m^{i\to p(i)} \leftarrow NN_{bu}(s_i, \{m^{c\to i}\}_{c\in\mathcal{C}(i)})$
22:     **end if**
23: **end for**
24: // top-down message passing (root to leaf)
25: **for** node $i$ in nodeList **do**
26:     **if** $i$ is root **then**
27:         $a_i, \{m^{i\to c}\}_{c\in\mathcal{C}(i)} \leftarrow NN_{td}(m^{i\to p(i)}, \vec{0})$
28:     **else**
29:         $a_i, \{m^{i\to c}\}_{c\in\mathcal{C}(i)} \leftarrow NN_{td}(m^{i\to p(i)}, m^{p(i)\to i})$
30:     **end if**
31: **end for**
32: **return** $\{a_i\}_{i=1}^k$
```

As shown in Figure 1, both the leaves' states and the actions are more correlated with the closer messages passed to them, demonstrating that messages are indeed conveying meaningful contextual information for locomotion.

A.4. Sanity Check Experiment

We perform a sanity check on the single-agent environment to see if a modular policy can learn as well as a monolithic one based on the message passing formulation discussed in Section 3 in the main paper. It can be seen from Figure

3 in the main paper that good coordination between limbs is of the utmost importance for control because under the no message setting, all limbs act as independent agents and cannot coordinate with each other and thus cannot learn anything meaningful. This shows that a modular policy is certainly at a disadvantage when only trained and tested on a single morphology against a monolithic policy since the latter is a much easier optimization problem and does not need to learn to generate messages just to coordinate limbs. In contrast, a modular policy has to learn message passing as well as the controller to generate meaningful behaviors. Despite the challenges, we observe that, as shown in Figure 2, the Shared Modular Policies with both-way message-passing can achieve comparable performance to a monolithic baseline in all experiments.

A.5. Pseudo-Code for Shared Modular Policies

We provide the pseudo-code for training Shared Modular Policies with both-way message passing. Algorithm 1 discusses the joint setup of sharing policies across motors of all agents, and Algorithm 2 discusses the end-to-end training.

References

Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., and Zhokhov, P. Openai baselines. `https://github.com/openai/baselines`, 2017. 1

Fujimoto, S., Van Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. *ICML*, 2018. 1

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. *ICLR*, 2015. 1

Polosukhin, I. and Zavershynskyi, M. nearai/torchfold: v0.1.0, Jun 2018. URL `https://doi.org/10.5281/zenodo.1299387`. 1