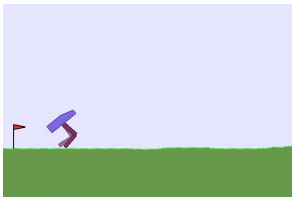
One Policy to Control Them All: Bipedal Environment

Niklas Dettmer, Marko Duda

March 2021

1 Introduction

(Huang, Mordatch, & Pathak,) show with their paper "One Policy to Control them All" that with Shared Modular Policies (SMP) - where actuators such as knees or hips of an agent are all sharing the same policy - can compete with usual, monolithic policies. (Huang et al.,) use Multi-Joint dynamics with Contact (*MuJoCo*,) to build their environments. We investigate whether SMP can be transferred to other movement-coordinating environments like the gym environment BipedalWalker (*BipedalWalker*,) by OpenAI (see picture below).



This question is not trivial. The state design of Bipedal Walker is very different from the MuJoCo environments described by (Huang et al.,). They provide each actuators' state with information such as global position of the actuators. The Bipedal Walker does not have these global assignments for its actuators. Its state is designed with a monolithic policy in mind. If we can get BipedalWalker off the ground with SMP, then this is a strong indication that the underlying concept of SMP is truly significant and general for Multi Agent Reinforcement Learning. In the next section, we describe the basic concept of SMP. Then, we describe the specifics of our implementation and the specifics of BipedalWalker. After that we compare the performance of BipedalWalker executed with SMP and with the basic monolithic approach.

2 Method

This section describes the concept behind Shared Modular Policies (SMP). For SMP, the agent is split into its actuators. Each actuator has its own state which is fed to the actuators' policy to generate the actuators' action. This makes SMP a multi agent algorithm. Each actuator is its own agent. Note however that we still mean the whole morphology - meaning the whole walker - when talking about the agent. What is special about SMP is that all actuators share the same policy. This approach alone is local and therefore not sufficient for sophisticated tasks like walking with two legs. Walking requires global coordination between the actuators. This is why the actuators of (Huang et al.,) communicate with their neighbor actuators in order to enable global coordination.

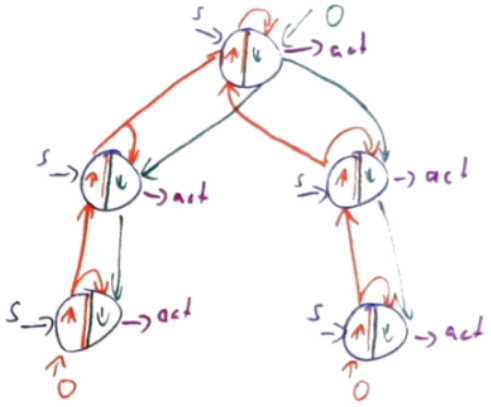


Figure 1: Morphology of BipedalWalker. The bottom nodes are the knees, the middle nodes are the hips and top node is the head or hull joint of BipedalWalker. Red are bottom-up and green are top-down messages.

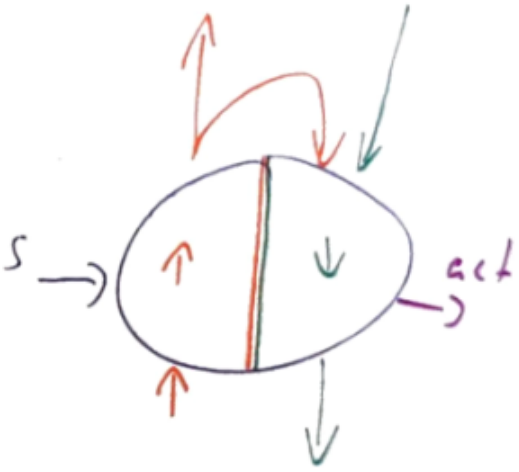


Figure 2: The left half of the circle is the bottom-up policy. It receives the previous message from below and generates the next one. The top-down policy, the right half of the circle, gets both the top-down as well as the bottom up message and generates the actuator's action and the next bottom up message.

(Huang et al.,) showed that the best performance is achieved when the actuators communicate in both directions as visualized by the red (upwards) and green(downwards) arrows in Figure 1.

The messages are produced sequentially starting with the bottom-up messages. The first messages are generated from the leaf nodes, the knees, and are passed to the next node, the hips. The hips nodes pass their messages to the the head. After this bottom-up pass, the top-down messages as well as the actions are generated by the top-down pass (visualized in green).

It might seem odd that the top-down policy, responsible for generating the actions, just receives messages but not the actual state as an input. By looking at the policy architecture from a different perspective, one can see that the bottom-up policy and the top-down policy are not as separated as one might think. The message the bottom-up policy produces are an ordinary layer output. Therefore, the bottom-up policy can be considered as the earlier layers of the top-down policy. (see Figure 3).

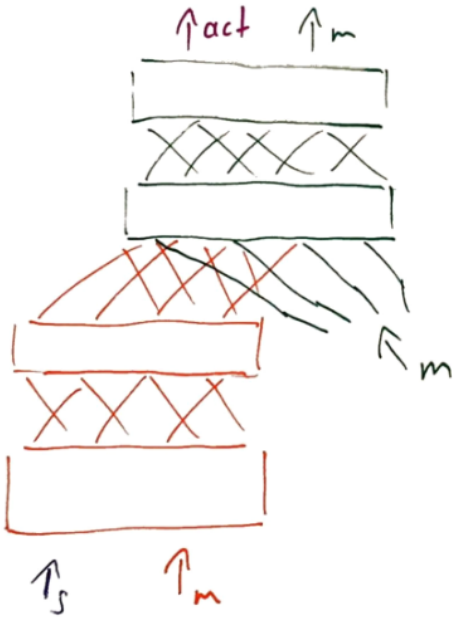


Figure 3: Another perspective at the architectures of both policies. The bottom-up policy (red) can be considered as the early layers of the top-down policy (green).

One interesting site note at the end of this section: Previously to SMP, (Huang et al.,) implemented someting similiar to SMP. In (Pathak, Lu, Darrell, Isola, & Efros,), they use a collection of primitive agents which learn to dynamically self-assemble themselves 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.

3 Implementation

SMP is based on TD3 (Fujimoto, van Hoof, & Meger,). So at first, we implemented the usual TD3 algorithm with the ReAlly framework. Next, we replaced the basic actor network with an SMP-Actor network. For that we had to split the global state into local states suited for each actuator. As already mentioned in the introduction, the state of BipedalWalker was designed with a monolithic policy in mind. This makes splitting it into its actuator states a non-trivial task.

3.1 TD3

We implemented the TD3[1] algorithm as a child class of the basic tensorflow modul class. This child class contains an actor module and two critic modules. The actor module outputs μ and σ of the action distribution. Within the optimization loop, we apply the all the techniques needed for TD3. We perform target policy smoothing by adding noise to the action fed to the critics to create the target, we apply double Q-learning - which is why we have two critic layers in the first place - and we delay the update of the target policy.

So far, we have a monolithic policy trained with TD3. A first experiment with this monolithic policy

[1] We expect the reader to know how TD3 works in general so we won't go into further detail concerning the general algorithm.

confirms that the agent learns something useful: It crawls over the ground.

3.2 Shared Modular Policies (SMP)

Next, we are going to replace the already implemented monolithic actor module by an SMP actor module. Note that, although possible, we do not replace the monolithic critic by an SMP critic (for the sake of simplicity). The shared modular policy technique and thus the transfer of the technique of (Huang et al.,) is realized by modularizing the states and actions within the actor module. That SMP actor module consists of a bottom-up network module and a top-down network module. The monolithic state of the actor is splits up into the five modular states for each joint inside the actor. Now node after node we perform the bottom-up messaging process by feeding preceding messages and corresponding current states to the bottom-up network module which creates upwards messages. Once we finished that process we perform the top-down process feeding created actions and corresponding downwards messages to the top-down network module which creates actions for the current actuator node and follow-up downwards messages. The initial bottom-up messages, which are the input for the knee nodes, and the action for the head joint are always filled with 0 for initialization.

Once we have finished the bottom-up and top-down messaging and created actions for all actuators we concatenate the actions back together in order create one monolithic action distribution from those modular action distributions. The API of the SMP actor module is the same as the one of the monolithic approach.

Both the bottom-up and the top-down network together from a single network that can be accessed from and trained by the optimization loop.

3.2.1 State and action design

In order to realize the modular approach we had to split up the environment states and actions into modular states and actions for each joint. For this we retrieved the meaning of the dimensions of the monolithic states and actions from the implementation of the BipedalWalker environment[2].

In the environment a state is formulated as $(\phi_{\text{hull}}, \Delta\phi_{\text{hull}}, v_x, v_y, \phi_{\text{hip}_l}, \Delta\phi_{\text{knee}_l}, g_l, \phi_{\text{hip}_r}, \Delta\phi_{\text{knee}_r}, g_r)$ where ϕ stands for the angle of an actuator, $\Delta\phi$ for the angle speed of an actuator, $\vec{v} = (v_x, v_y)^T$ for the 2D velocity of the walker's centre of gravity, g for boolean variables which indicate if the corresponding leg currently has ground contact, and the subscripts l and r indicate the side of the leg: left or right.

We transformed that state into modular states formulated as $(\phi, \Delta\phi, g, v_x, v_y)$ with $g = 0$ for each non-knee actuator and $\vec{v} = 0$ for each non-head actuator. So only the knee actuators sense ground contact and only the head senses the velocity of the whole walker. Thus the knee actuators have some kind of feet-sensation and the head actuator has some kind of a vestibular system. The approach to always choose the largest state and action dimension and zero out the unknown channels was also used by (Huang et al.,).

An action in the environment is formulated as $(\delta\phi_{\text{hip}_l}, \delta\phi_{\text{knee}_l}, \delta\phi_{\text{hip}_r}, \delta\phi_{\text{knee}_r})$ which are just the forces applied to each actuator. Such an action is easily created by combining the obtained 1D

[2] github.com/openai

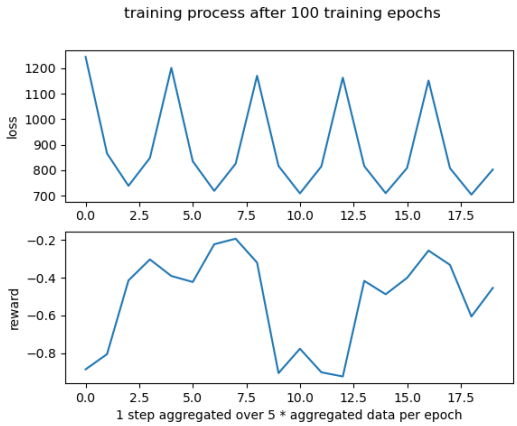


Figure 4: Progress of baseline run

actions from the corresponding actuator's policy networks.

The original states provided by the gym environment also contain 10 lidar entries. The lidar is some kind of a laser scanner mounted to the head joint of the walker. We ignored that information in most of our runs to be more close to the environment of (Huang et al.,).

4 Results

Using a simple grid search we obtained some promising hyper parameter values which we used in four final training procedures. In these training procedures we did some further experiments.

4.1 Grid Search

To find the best working hyper parameters we applied a simple form of a grid search where we just tried out all combinations out of a set of parameter values. We obtained the highest rewards for a run with a discount factor of 0.98, a learning rate of 0.0004, a message dimension of 16 channels, and a policy noise with $\sigma = 0.5$.

4.2 Final Runs

In the end we managed to perform 3 different training procedures using shared modular policies. All of them had the configuration which had the highest rewards in our grid search (see previous section). We did some experiments with a larger buffer, a longer training process and putting the lidar (see 3.2.1) into the state of the head joint. Unfortunately neither in our baseline nor in our SMP runs we achieved a reward > 0. We will talk about this in more detail in 5.

In our baseline where we just applied TD3 using a monolithic policy model. In this run the walker always freezes having the legs straight in an almost 90° angle from each other with some small differences which make the walker do a front flip sometimes. The loss changed periodically as the reward stays slightly below 0. The best reward was ≈ 0.18.

The first SMP run used a relatively small buffer size of 10,000 and the hyper parameter values

Figure 5: First SMP run with small buffer size

Figure 6: Second SMP run with bigger buffer

obtained from our grid search. Here we see that the model has learned to freeze the legs in an acute angle to each other which makes the walker slowly fall forward as the head tips over to the front. Again we have a periodically changing loss with a reward slightly below 0. The best reward was ≈ 0.3.

In our second SMP run we increased the buffer size to 100,000. The model learned to put one leg straight and bend the other in a 90° angle. This makes the walker fall slowly backwards most of the time, but sometimes it jumps in the air and performs a front flip. The loss again changes periodically while the reward fluctuates slightly below 0, but in the end it falls steeply to below -2.

In our third and final SMP run we added the lidar state information to the state of our head joint. Now almost each joint gets freezed again, but one of the knee actuators wiggles. The walker slowly falls over backwards again. The loss changes periodically in a relatively small interval of approximately 200 units. The reward in the end stays between -0.5 and -1.5.

5 Evaluation

(Huang et al.,) showed that SMP approaches can only be noticeably superior to monolithic approaches when being trained on various morphologies which is depicted in Fig. 8. Nevertheless they also show that modular approaches can keep up with monolithic ones when being trained on a single

Figure 7: Third SMP run with lidar information

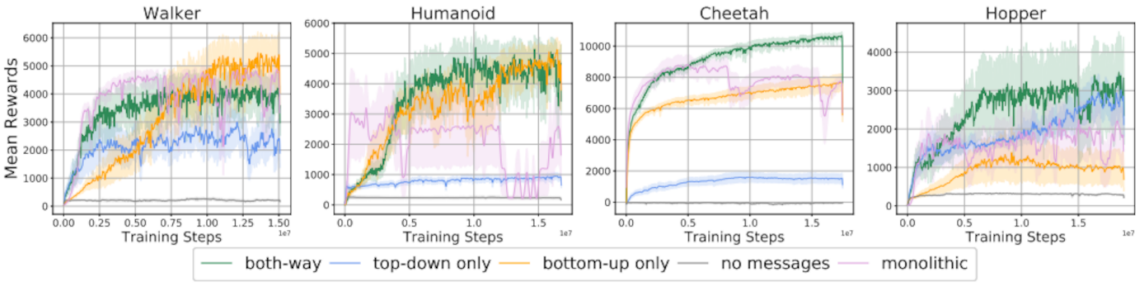


Figure 8: In this "sanity check" (Huang et al.,) trained their SMP models on a single morphology.

morphology as in our experiment.

As the loss functions always stay within an interval and change exactly periodically, we believe that the models always get stuck in local optima. These optima seem to be episodes in which the walker manages not to fall for as much time as possible. A good general solution for this seems to be to just freeze the actuators in one position and avoid big moves then. We found the third SMP run to be the most interesting, because we see the walker trying to keep its balance by wiggling with one joint. All in all one can argue that the walker learns to stay above the ground for as much time as possible. This is achieved pretty well in both the monolithic baseline and the SMP approaches. We see that SMP approaches and usual monolithic approaches are indeed able to keep up when trained on a single agent morphology.

6 Outlook

A main problem of ours was our constraints in hardware and time. In their implementation (Huang et al.,) chose hyper parameters which clearly exceeded our hardware resources. With a larger buffer, larger samples and more training time we might have been able to actually get the walker to walk. Nevertheless we managed to teach the walker to stay above the ground for as much time as possible.

For further exploration one could run our code on a more powerful machine with way more samples. One could also do a way more complex grid search to find optimal hyper parameters.

We also thought about doing a domain randomization by randomizing environment variables like the ground friction. Doing this could be have a similar effect as training the model on different

morphologies.

All in all we are proud to present our results.

References

Bipedalwalker. (n.d.). https://gym.openai.com/envs/BipedalWalker-v2/. (Accessed: 2021-04-12)

Fujimoto, S., van Hoof, H., Meger, D. (2018, feb). Addressing Function Approximation Error in Actor-Critic Methods. *35th International Conference on Machine Learning, ICML 2018*, *4*, 2587–2601. Retrieved from http://arxiv.org/abs/1802.09477

Huang, W., Mordatch, I., Pathak, D. (2020). *One Policy to Control Them All: Shared Modular Policies for Agent-Agnostic Control* (Tech. Rep.). Retrieved from https://huangwl18.github.io/modular-rl/.

Mujoco. (n.d.). http://www.mujoco.org/.

Pathak, D., Lu, C., Darrell, T., Isola, P., Efros, A. A. (2019, feb). Learning to Control Self-Assembling Morphologies: A Study of Generalization via Modularity. *arXiv*. Retrieved from http://arxiv.org/abs/1902.05546