Soft Shape-Shifting Robots: Is there a better way?

Miriam Bakija, Laura Ladaru

April 2024

1 Abstract

Recently Huang et al [3] created a new open benchmarking environment to test soft morphology changing robots. In their introduction paper to the new environment they compared the performance of different network architectures on different soft robot morphology tasks. To compare all of these they used the Soft Actor Critic (SAC) [2] an off-policy reinforcement learning algorithm. We explore the performance of two on-policy reinforcement learning algorithms in this environment: Proximal Policy Optimization (PPO) [9] and Advantage Actor Critic (A2C) [6].

2 Introduction

Co-design – where both the physical body and control policy of a robot can be optimized together for a task – is a hot area of research in robotics. Soft robotics is particularly well-suited to new manufacturing methods that can realize adaptable morphologies and actuators. A recent development in this area are reconfigurable robots, whose morphology can change dynamically. Recently the first benchmark environment for soft-robots that actively change their morphology was created under the name Dittogym [3].

Our project aims to explore how different Reinforcement Learning (RL) algorithms perform in the context of controlling reconfigurable soft robots. Previously, Huang et al. [3], benchmarked several different neural network architectures for learning optimal policies for reconfiguration and action on several different soft robot morphology tasks. All of these different architectures were benchmarked using the off-policy Soft Actor Critic (SAC) algorithm. Our work aims to learn more about if on-policy RL algorithms can succeed in this context. Specifically, we examine how PPO and A2C perform on different tasks, as well as what potential issues can arise when implementing these algorithms in such an environment.

By rigorously assessing the reproducibility and generalizability of these RL algorithms in controlling reconfigurable soft robots, we aim to advance our understanding of their capabilities and limitations. Additionally, we aim to confirm the suitability of SAC for this task, considering its inherent advantages in handling continuous action spaces and complex environments.

A2C and PPO are both powerful reinforcement learning algorithms, but they have limitations in certain scenarios compared to SAC. We find that SAC is indeed an optimal choice of base RL algorithm for the reconfigurable soft-robots.

3 Related work

Previous research, such as Luck et.al. [5], has examined the adaptation of robot morphology and behavior using the Soft Actor-Critic (SAC) algorithm. They chose SAC due to its compatibility because it well suited to work in the context of deep learning and its data efficient approach of sampling old morphologies.

Another very relevant paper to our work is Kalimuthu et.al. [4] who compared the performance of PPO and A3C in adapting robot morphologies specifically for the Smorphi bot. They found that PPO not only was more energy efficient (the main focus of the study), but more consistent in finding increasingly better morphologies. The results from this paper are part of what motivated us to choose PPO over different on-policy policy gradient methods such as REINFORCE or Trust Region Policy Optimization [7], because PPO had been shown empirically to work in a similar context.

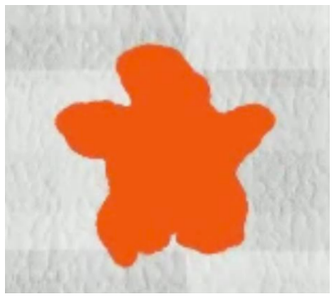

Figure 1: Shape Match Task

4 Environment Details

Soft robots within the Dittogym environment are simulated using the Material Point Method, a particle-based approach that replicates a continuous muscle field akin to magnetic slime, as detailed by Sun et al. (2022) [10]. To facilitate task learning, this continuous muscle field is transformed into a 2D action space, upon which reinforcement learning algorithms are applied to determine optimal actions for the muscle field.

Huang et al. benchmarked the performance of several policy network architectures within this environment. Notably, their course-to-fine architecture emerged as the most effective. Course-to-find is built upon the foundational course architecture, which also demonstrates robust performance in benchmarks. The course architecture involves first training an observation encoder to capture an ideal representation of the state. Then, a Gaussian policy network selects actions based on this encoding, optimizing both the morphology and the actions of the soft robots. We employ the course structure to learn a policy in our methodology.

The dittogym environment provides eight different tasks that test the soft robots algorithm to learn both actions and a morphology to succeed. In our paper we focus on 3 tasks: Shape match, Kick and Dig. We chose to focus on these tasks because of their relatively shorter horizon to learn. The shape match task tests the robot's ability to change its morphology into the shape of a star. An example of this can be seen

in figure 1. Shape match is considered the easiest task in the set, as it is simulated with the softer material, making locomotion easier. The next task we examine is Kick, where the goal is to move an object as far as possible. Kick is more difficult due the task complexity and simulated with a harder material. Finally the Dig task aims to move the robot to dig through soil to reach a block. This task has similar difficulty to Kick, though the learning horizon is longer.

5 Methodology and Implementation Details

The code for the Course network architectures and soft actor critic is publicly available on GitHub [3] and served as a starting point for our implementations of the two algorithms, though major changes needed to be made for our algorithms to work properly due to problems we will discuss. We implemented our base RL algorithms and changed the *train.py* script with specific training code for each of the SAC, PPO, and A2C. To follow the style of the SAC implementation, we created an "on-policy memory buffer" in order to support our on-policy algorithms. Then. we test our algorithms on the *dittogym* tasks described above and utilize the WANDB platform (Weights and Biases) to view live charts of the performance metrics logged during training.

5.1 Soft Actor Critic

We used the SAC implementation done by Huang et.al. [3], which follows a standard implementation of SAC, but with some notable details. First, two critics are trained simultaneously, a method called SAC-Lagrangian [1]. This method adds additional complexity and stabilization of the learning process, potentially giving SAC an advantage over other algorithms. Additionally, while entropy-tuning with a decreasing alpha was implemented, they did not turn it on in benchmark tests. In contrast our implementations both use entropy regularization.

5.2 Proximal Policy Optimization implementation

We implemented PPO-clip which takes the ratio of the in log probability from the old policy and the current policy and clips the ratio so no extreme points will change the policy too much. We also created a Value neural network with similar structure to that of the Q-network, as PPO learns a value function and not a Q function. An important distinction is that while both the Q-Network and V-Network use the same observation encoder, the Q-network has the ability to merge action and state into one observation input, obviously this is not needed in the Q-Network. One issue we repeatedly encountered when trying to implement our on-policy algorithms was positive log probabilities from the policy network, which was due to small standard deviations that did not change much with policy updates due to an issue with clamping. To solve this issue we had to come up with some creative solutions, including changing std layers from ELU to leaky ReLU, increasing the *min_std* and adding the *min_std* before clamping, in order to increase diversity in std predictions. While this mostly fixed the issues we encountered with positive log probabilities, we still occasionally had issues with getting 10,000 steps in or further and encountering positive log probabilities something we will discuss further in the challenges and limitations section.

Another important part of our implementation was regularization of the policy loss function with entropy. We found that this was essential to the exploration of PPO and found that no implementation would learn correctly without it.

We chose optimal hyperparameters by using a guess and check method, intuitively changing based on if we observed the algorithm needed more exploration or exploitation by graphing metrics like entropy, log probabilities, and training step reward. For details on ideal hyperparameters for PPO see appendix table 1.

5.3 A2C implementation

Our implementation of A2C closely follows SAC, with a few changes. Unlike SAC where there is a soft update step, in A2C, the objective is to maximize the expected return (calculated using Monte Carlo) by directly optimizing the policy and value function. The policy is trained to increase the probability of actions that lead to higher returns, while the value function is trained to predict the expected return from a given state.

The A2C class initializes the agent with various parameters and neural network models for the policy and value function. It sets up optimizers for both the policy and value function networks, and if automatic entropy tuning is enabled, it sets up the parameters for entropy adjustment. The action selection method, for selecting an action based on the current state, takes the state as input and returns the action, along with other information such as the log probability, mean, and standard deviation of the action distribution. The *update_paramateres* method samples batches of experiences from memory and performs updates on both the policy and value function networks. It calculates advantages using Monte Carlo returns and updates the critic (value function) using the mean squared error loss between predicted values and observed returns.

Building on top of the mentioned issues in Section 5.1 regarding the positive log probabilities, even with multiple attempts at clamping the *log_pi* values, tuning hyperparameters, normalizing rewards and advantages, or trying different computation approaches, we could not fully escape the increasing positive *log_pi* values in A2C. Just as well, given that A2C is a more basic approach than SAC, some differences between the implementations lowered the quality of results and effectiveness. A2C relies on entropy regularization to encourage exploration, but SAC directly incorporates entropy into its objective function, allowing for more principled exploration strategies. A2C updates the critic (value function) using a mean squared error loss between predicted values and observed returns, but SAC updates the critic using a Bellman error derived from the temporal difference between current and target Q-values. The Bellman error used in SAC provides a more stable update signal compared to the mean squared error.

While we do use a memory buffer, we implemented an on-policy buffer in order to compute the returns and actively update the policies on the current state, so our algorithms remain on-policy approaches. However, SAC is an off-policy algorithm, meaning it can learn from past experiences stored in a replay buffer. This allows for more efficient use of data and can lead to faster convergence compared to on-policy algorithms like A2C and PPO.

6 Results

6.1 Proximal Policy Optimization

We tested PPO on all four tasks, with varying degrees of success. First on the easiest task shapematch we did observe the robot learning, though at a slower rate than the benchmark SAC, the results of which can be observed in figure 2. We found for shape match the ideal hyperparameters listed in the appendix with only deviations being decreasing initial noise for mean and std weights down from 0.1 to 0.03. The kick task provided the most similar learning pattern between PPO and SAC, as seen in figure 3. In terms of deviations from optimal hyperparameters for Kick for PPO, we found that performance improved with a batch size of 400, decreasing the policy optimizer learning rate to 0.0001, and increasing the entropy coefficient to 1.5. Finally, we attempted to train using PPO on the Dig task, the results of which can be seen in figure 4. We did not observe any learning in either SAC or PPO, though this is likely due to the fact of the long episode length, which made training take an extensive amount of time, especially for SAC which updates once per step.

Overall, we found it to be somewhat difficult to learn using PPO and compared to SAC. One concerning problem with PPO and A2C we ran into often especially with the shape match task was log probabilities steadily increasing into positive numbers, which lead to errors in gradient updates ending the training run prematurely. We suspect that this was related to very little change in the average entropy over time, and were able to thwart this problem using strategies such as adding noise to the initial std weights and decreasing the number of updates per episode, however occasionally the problem still arose. If more time could be spent on hyperparameter tuning, we think that the issue could be significantly improved, however this issue also may be somewhat inherent to implementing on-policy algorithms in these complex environments, without finding complex solutions. But as seen in figures 2 and 3 it is possible for soft robots to learn under an on-policy algorithm.

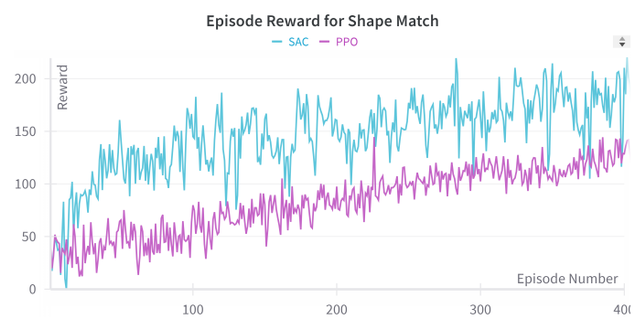

Figure 2: Shape Match

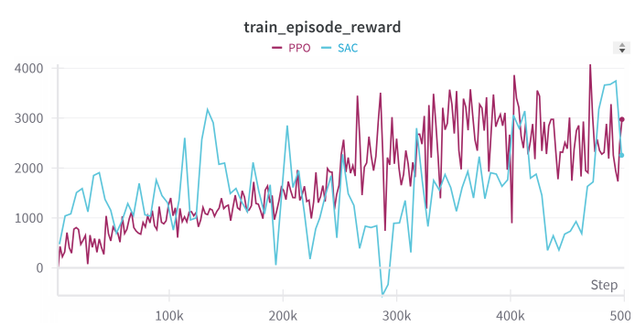

Figure 3: Kick task

Figure 4: Dig task

6.2 Advantage Actor Critic

We observed a great performance disparity between A2C and PPO/SAC in achieving high rewards. As seen below, A2C struggles to learn and gain and keep high rewards even with multiple attempts at hyperparameter-tuning and these are the best attempts with $learn\_rate = 3e-8$ for both policy and critic, $batch\_size = 256$ and the rest being default hyperparameters, a complete list being available in *Appendix Table 2*. The learn rate was drastically lowered due to numerical issues that caused the learning to stagnate and even fail at times. This can be attributed to several key factors. A2C's reliance on episode-based updates, compared to the more sophisticated experience replay mechanisms of PPO and SAC, suggests a potential limitation in sample efficiency. This could result in a narrower exploration of the state-action space, hindering the discovery of optimal policies.

The frequency of policy updates in A2C, occurring after each episode, contribute to slower convergence rates compared to the more frequent updates utilized in PPO and SAC. This can be easily seen in Figure 5, where the agent seems to very slowly learn the Dig task, based on the slight curving shape of the rewards plot. To further consolidate the slow convergence, Figure 7 demonstrates that the agent is actually learning, given by the final action at the moment of the manual stop of the execution, which changes from the initial final task.

Figure 5: Dig Task A2C and SAC

Figure 6: Kick Task A2C and SAC

Figure 7: In order: Initial (step 0), Dig, Kick Task - Final action visual

Moreover, we believe the variance in A2C's policy gradient estimates (*log_pi* issue mentioned above), especially in continuous action spaces, led to unstable training and slower convergence. Sensitivity to hyperparameters, such as learning rate and entropy coefficient, complicated its performance optimization, since finding optimal values proved to be very difficult.

Unfortunately, we were unable to run the task Shapematch on A2C due to unknown issues regarding Illegal memory access using CUDA, which we were unable to solve for this algorithm.

Thus, using A2C for this environment and especially for the Dig and Kick tasks is not an appropriate choice due to slow convergence, low rewards, and numerical instability. While A2C serves as a fundamental RL algorithm, we do not believe it is suitable for addressing the challenges posed by the reconfigurable soft robots. Future research efforts may explore alternative RL approaches, such as the chosen SAC, that offer more robust solutions for navigating complex and dynamic environments.

7 Challenges and limitations

Because of access to limited resources in terms of GPUs, we were not able to run algorithms for nearly the same number of steps as done in the benchmark paper. However we feel this does not completely neuter the conclusions we draw from the results as performance changes in the RL algorithms develop fairly early on at around on average 20,000 steps in. Additionally are results lack a certain level of robustness because we could only feasibly do one run on a set of hyperparameters. Ideally, we would have done multiple runs on the same hyperparameters, but we felt that out time was better spent choosing optimal hyperparameters that would lead the algorithms actually learning.

Also because SAC updates once per step running a single episode took much longer in comparison to PPO and AC2 which only updates at the end of an episode because it must update based on return. This made comparisons in performance between SAC and PPO difficult to see as the number of updates to episodes ratio is much higher with SAC so it appears that PPO is a slower learner when graphed over episodes even though the amount of real world time to learn is the same.

8 Discussion and Conclusion

There are many important lessons to be taken away from this project.

Complexity: The task of controlling reconfigurable soft robots involves navigating a high-dimensional action space that encompasses both morphological changes and locomotion strategies. A2C and PPO's reliance on Monte Carlo returns may struggle to efficiently explore and exploit this intricate action space, particularly when fine-grained control is required to achieve task objectives. Future experimentation could be done to see if implementing Generalized Advantage Estimation [8], a more complex return calculation, would improve the performance.

Exploration vs Exploitation: PPO and A2C's exploration strategy, primarily driven by entropy regularization, may face challenges in striking an effective balance between exploration and exploitation in environments with complex dynamics and morphology changes. In contrast, the SAC algorithm offers a more principled approach to managing this trade-off through its direct incorporation of entropy into the objective function. Our efforts to improve exploration included adding noise to the initial weights of the mean and std Gaussian policy, and significantly increasing the coefficient on entropy regularization but this only marginally improved exploration and not to the level that SAC had.

On vs Off Policy: Our algorithms' on-policy nature limits its ability to efficiently utilize data, particularly in scenarios where past experiences could inform learning. SAC, as an off-policy algorithm capable of leveraging past experiences stored in a replay buffer, seems to demonstrate far better sample efficiency and convergence speed compared to A2C in the context of co-designing reconfigurable soft robots.

A future line of investigation could be to explore modifications or hybrid approaches that combine elements of both on-policy and off-policy algorithms. For example, we could investigate hybrid algorithms that leverage the advantages of both A2C and SAC or incorporate ideas from other algorithms like DDPG (Deep Deterministic Policy Gradient). Also, due to the time constraints, it would be worth revisiting A2C's reward function and experiment with different reward shaping techniques. This could help address the issue of A2C struggling to achieve high rewards.

In conclusion, our investigation into the performance of PPO and A2C algorithms compared to the benchmark the SAC algorithm to control reconfigurable soft robots revealed important insights. While PPO and A2C possess their own strengths, particularly in simpler environments, our findings underscore SAC's superior sample efficiency, convergence speed, and stability when navigating the complex dynamics of these robots. Challenges encountered, including issues with positive log probabilities and slow convergence rates in A2C, highlight the need for algorithmic suitability and efficiency in order to address tasks such as these. Our study suggests the potential for hybrid RL algorithms and alternative reward shaping techniques to further improve performance in this domain. We are grateful to have learned about such a new and fascinating area of study, and we hope to be able to further contribute to it ourselves in the future.

9 Statement of Contributions

Miriam Bakija implemented PPO and Laura Ladaru implemented A2C. Both authors ran the experiments, wrote the paper and made the video.

References

[1] Sehoon Ha, Peng Xu, Zhenyu Tan, Sergey Levine, and Jie Tan. Learning to walk in the real world with minimal human effort. *arXiv preprint arXiv:2002.08550*, 2020.

[2] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[3] Suning Huang, Boyuan Chen, Huazhe Xu, and Vincent Sitzmann. Dittogym: Learning to control soft shape-shifting robots. *arXiv preprint arXiv:2401.13231*, 2024.

[4] Manivannan Kalimuthu, Abdullah Aamir Hayat, Thejus Pathmakumar, Mohan Rajesh Elara, and Kristin Lee Wood. A deep reinforcement learning approach to optimal morphologies generation in reconfigurable tiling robots. *Mathematics*, 11(18):3893, 2023.

[5] Kevin Sebastian Luck, Heni Ben Amor, and Roberto Calandra. Data-efficient co-adaptation of morphology and behaviour with deep reinforcement learning. In *Conference on Robot Learning*, pages 854–869. PMLR, 2020.

[6] Baolin Peng, Xiujun Li, Jianfeng Gao, Jingjing Liu, Yun-Nung Chen, and Kam-Fai Wong. Adversarial advantage actor-critic model for task-completion dialogue policy learning. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6149–6153. IEEE, 2018.

[7] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[8] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[9] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[10] Mengmeng Sun, Chenyao Tian, Liyang Mao, Xianghe Meng, Xingjian Shen, Bo Hao, Xin Wang, Hui Xie, and Li Zhang. Reconfigurable magnetic slime robot: deformation, adaptability, and multifunction. *Advanced Functional Materials*, 32(26):2112508, 2022.

Appendix

Table 1: PPO Hyperparameter Details

Hyperparameter	**Value**
γ , Discount factor	0.95
β , Entropy regularization coefficient	1.0
α_1 , Policy optimizer learning rate	0.0003
α_2 , Value function optimizer learning rate	0.003
Gradient clipping (Policy)	10
Gradient clipping (Value)	100
L2 regularization ,Policy optimizer weight decay	0.0001
ϵ PPO clip	0.2
Noise (Initial mean parameters)	0.1
Noise (Initial std parameters)	0.1
Batch Size	202
Number of updates per episode	5
Steps before update	1000

Table 2: A2C Hyperparameter Details

Hyperparameter	**Value**
γ , Discount factor	0.99
β , Entropy regularization coefficient	3.0
α_1 , Policy optimizer learning rate	3e-8
α_2 , Value function optimizer learning rate	3e-8
Gradient clipping (Policy)	5
Gradient clipping (Value)	10
L2 regularization ,Policy optimizer weight decay	1e-2
ϵ PPO clip	N/A
Noise (Initial mean parameters)	0.1
Noise (Initial std parameters)	0.1
Batch Size	256
Number of updates per episode	1
Steps before update	1000