Optimize Robots Physical Design by Parameterization in Deep Reinforcement Learning

Peide Huang
Department of Mechanical Engineering
Stanford University
`pdhuang@stanford.edu`

Abstract

When designing robots, it is a common practice to decide some mechanical parameters prior to carrying out any detailed design. However, it is often difficult to find the optimal values for those parameters. In most of reinforcement learning tasks, the agent is learning a control policy in a fixed environment. This project aims to enable the agent to modify some environment parameters related to its physical configuration, i.e. the agent is able to evolve to a better version that is more suitable for a certain task. We proposed methods to argument the reward function to influence the direction of evolution according to our preference. We discovered how the joint learning of policy and optimal physical configuration will impact the learning efficiency. We also found some interesting relationship between the learning of control policy and the evolution of the physical configuration.

1 Introduction

Doing mechanical design for robots is more than doing CAD in SolidWorks. Oftentimes it involves many iterations of design-test-design. However, it may become a problem when the designer want to figure out some essential mechanical properties prior to carry out any design because iterations can be expensive and time-consuming. Those mechanical properties can be the position of center of gravity, the stroke of actuator, etc. These properties are usually closely related to the specific task for which the robot is designed.

Bipedal robot is a classical topic in robotics and has been studied since late 1960s [1]. Researcher has conducted numerous studies in the physics of bipedal locomotion, designed many bipedal robot and developed some sophisticated control algorithms for it [2]. Since late 1990s, people started to introduce reinforcement learning to tackle this control problem [3, 4, 5]. After 2010s, the burst of deep learning ignited the usage of deep reinforcement learning in this field [6, 7, 8].

In most of the deep reinforcement tasks, the objective is to learn a policy that achieve a certain task. Usually, the agent is fixed. However, during the learning of control policy, if we allow the agent to modify some environment parameters, the agent may evolve to a better version of itself which is more suitable for the task. That is, our new parameters to learn is what the original network has plus the environment parameters, such as the length and weight of legs or the hull.

2 Related work

There is a wide range of literature in evolutionary computation that focus on modelling embodied cognition[9, 10, 11]. Theo Jansen [12] used evolutionary computation to design physical Strandbeests that can walk on their own using only wind energy. In the recent reinforcement learning researches, Agrawal et al.[13] used CMA-ES [14] to learn both the control policy and physical configuration of agents. More recent work [15] proposed a method to learn both alternately. Our work is based on the method proposed by Ha[16]. In this paper proposed a minor change to the rollout function of the original framework.

```
def rollout(agent, env):
  obs = env.reset()
  done = False
  cumulative_reward = 0
  while not done:
    a = agent.action(obs)
    obs, r, done = env.step(a)
    cumulative_reward += r
  return cumulative_reward
```

```
def rollout(agent, env_params, env):
  env.augment(env_params)
  obs = env.reset()
  done = False
  cumulative_reward = 0
  while not done:
    a = agent.action(obs)
    obs, r, done = env.step(a)
    r = augment_reward(r, env_params)
    cumulative_reward += r
  return cumulative_reward
```

Figure 1: Rollout change proposed by [16]

3 Environment

To provide a standard tool for developing and benchmarking the performance of reinforcement learning algorithm, OpenAI Gym library [17] collects many test problems(environments) and those environments have shared interface. One environment is called BipedalWalker-v2, which is a emulator of a bipedal robot that has 2 legs and each leg has 2 joints. The robot needs to learn a policy that applied torque on the 4 joints in order to move forward on a slightly uneven ground. This environment has been used in some researches and some algorithms has achieved great performance. The graphical rendering of the environment of BipedalWalker-v2 is as shown in Figure 2.

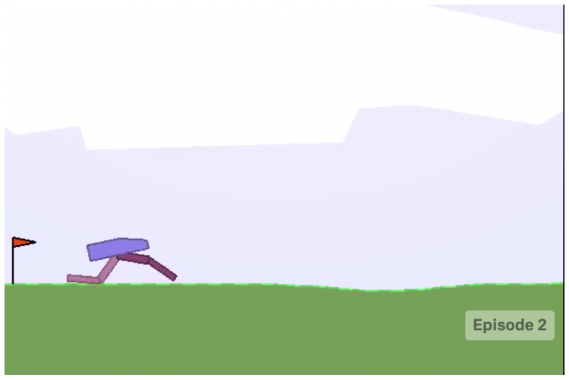


Figure 2: BipedalWalker-v2 environment

The observation space and action space is as shown below in Figure 3[17].

4 Methods

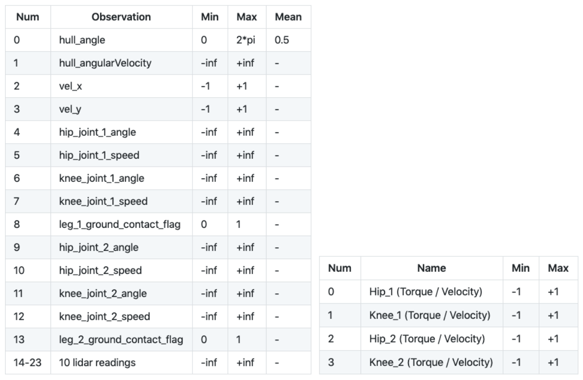
We used a simple two hidden layer fully connected neural network to learn the control policy. The input space layer size is 24. The output layer size is 4. Each hidden layer has 40 perceptrons.

In order to learn a parameter vector ω, we use Population-base Policy Gradient Method. ω is sampled from a probability distribution $\pi(\omega, \theta)$, a factored multi-variate normal distribution. The expected cumulative reward R is defined as:

$$J(\theta) = E_\theta\left[R(\omega)\right] = \int R(\omega)\pi(\omega, \theta)d\omega$$

Use the log-likelihood trick to write the gradient of $J(\theta)$ w.r.t θ as:

$$\nabla_\theta J(\theta) = E_\theta\left[R(\omega)\nabla_\theta \log \pi(\omega, \theta)\right]$$



Num	Observation	Min	Max	Mean
0	hull_angle	0	2*pi	0.5
1	hull_angularVelocity	-inf	+inf	-
2	vel_x	-1	+1	-
3	vel_y	-1	+1	-
4	hip_joint_1_angle	-inf	+inf	-
5	hip_joint_1_speed	-inf	+inf	-
6	knee_joint_1_angle	-inf	+inf	-
7	knee_joint_1_speed	-inf	+inf	-
8	leg_1_ground_contact_flag	0	1	-
9	hip_joint_2_angle	-inf	+inf	-
10	hip_joint_2_speed	-inf	+inf	-
11	knee_joint_2_angle	-inf	+inf	-
12	knee_joint_2_speed	-inf	+inf	-
13	leg_2_ground_contact_flag	0	1	-
14-23	10 lidar readings	-inf	+inf	-

Num	Name	Min	Max
0	Hip_1 (Torque / Velocity)	-1	+1
1	Knee_1 (Torque / Velocity)	-1	+1
2	Hip_2 (Torque / Velocity)	-1	+1
3	Knee_2 (Torque / Velocity)	-1	+1

Figure 3: Left: observation, right: action

Assuming that we have a population of N, we have parameters ω^1, ω^2, ..., ω^N, we can estimate the value as:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} R(\omega^i) \nabla_\theta \log \pi(\omega^i, \theta)$$

Then we can apply gradient ascent:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

We used the method described in William's paper[18] to calculate the closed-form formulas for calculating $\nabla_\theta \log \pi(\omega^i, \theta)$.

To influence the evolution of agent by modifying the reward function, we scale the rewards by a utility factor. For example, if we want to keep the bipedal walker symmetric, i.e. the left and right leg should be almost the same. We can calculate utility factor as shown below

Algorithm 1 Compute Reward Factor For Symmetric Legs

```
Require: vectors $v_l$ and $v_2$ representing the left and right leg parameters, sensitivity factor $\rho$
Ensure: reward factor $f$
1: function ComputeRewardFactor($v_1$, $v_2$, $\rho$)
2:     $d \leftarrow v_1 - v_2$
3:     $d_{scaled} \leftarrow \rho * L2norm(elementwiseDivision(d, v_1))$
4:     $f = 1 + log(1/(1 + d_{scaled})$
5:     return $f$
```

5 Experiments and Results

In this project, our experiments were conducted with a 96-CPU virtual machine on Amazon Web Service. We experimented with 4 ways of augmenting the reward function. As a baseline, we allow the agent to evolve freely without imposing constraints. Then we tested 3 different augmentations: balance(symmetric), small and tall legs. After training, the agent evolve to the configurations as shown in Figure 4. We can tell that the agent indeed evolved according to our expectation. During our experiments, although the evolution is in the direction we expected, the same augmentation could results in different final configurations. It means that the evolution is stochastic and subject to the random initialization of the environment.

We plot the training process in Figure 5. From this plot, we found that adding constraints to evolution will make the learning process less efficient compared with free evolution, although all of them converge and solve the task in the end.

From the plot, we found that there is a long flat region at the beginning of the training process. One of our hypothesises is that the agent is trying to find the optimal configuration before learning the control policy. To provide more evidence to this hypothesis, in Figure 6 we plot the training process

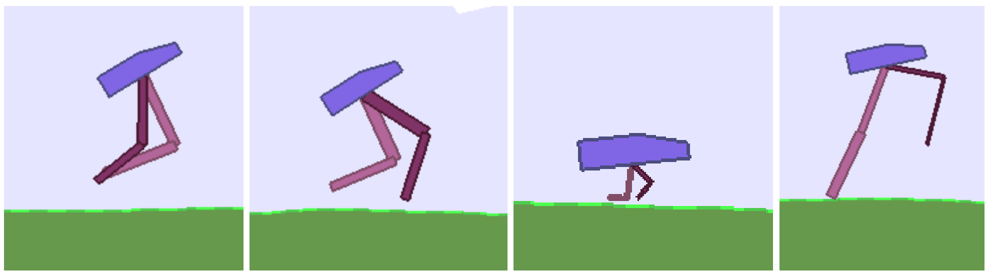


Figure 4: Final configuration: a) free evolution, b) balance, c) small, d) tall

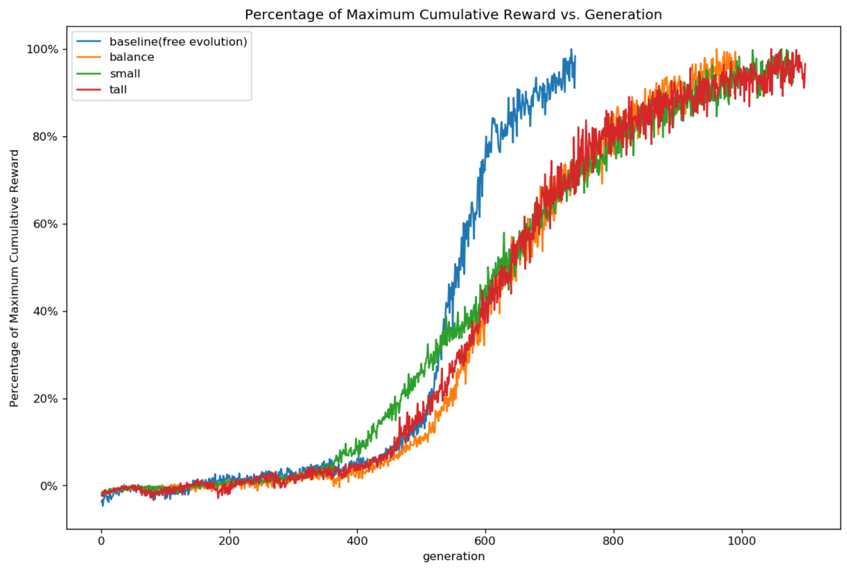


Figure 5: Training process comparison between different augmentations

and the evolution speed, defined as the L2 norm of the difference of body parameter vector between generations. We observed that the evolution speed tends to decrease over time. The improvement of cumulative reward starts to accelerate when the agent almost settles down with its physical configurations.

6 Conclusion

In this project, we explored different reward augmentations to influence the evolution of the agent's configuration. We discovered that constraining the evolution make the learning process less efficient than free evolution. We also discovered that the agent tries to evolve to a better version before it can start to learn the optimal control policy efficiently.

For future works, we can try deeper and more sophisticated neural network architectures to accelerate the learning process. In addition, we can also use different optimizers to update the environment parameters to improve the stability and convergence speed. After this, we should explore the possible generalization to tasks other than the bipedal walker.

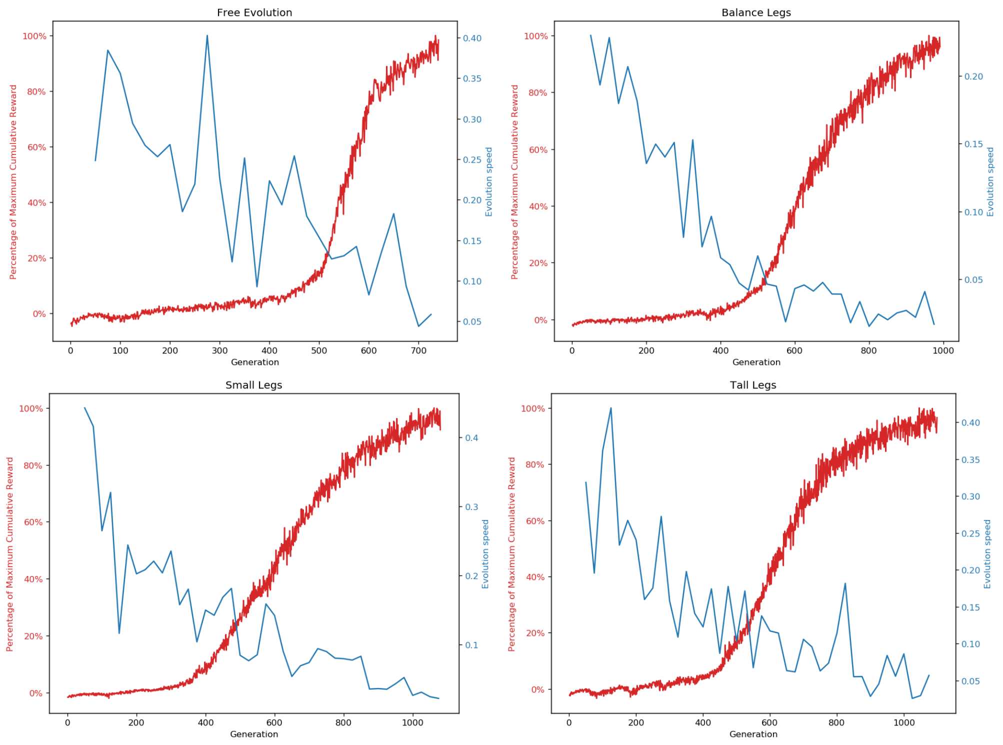


Figure 6: Cumulative reward and evolution speed: a) free evolution, b) balance, c) small, d) tall

References

[1] Aaron M Dollar and Hugh Herr. Lower extremity exoskeletons and active orthoses: challenges and state-of-the-art. *IEEE Transactions on robotics*, 24(1):144–158, 2008.

[2] Eric R Westervelt, Jessy W Grizzle, Christine Chevallereau, Jun Ho Choi, and Benjamin Morris. *Feedback control of dynamic bipedal robot locomotion*. CRC press, 2018.

[3] Hamid Benbrahim and Judy A Franklin. Biped dynamic walking using reinforcement learning. *Robotics and Autonomous Systems*, 22(3-4):283–302, 1997.

[4] Russ Tedrake, Teresa Weirui Zhang, and H Sebastian Seung. Stochastic policy gradient reinforcement learning on a simple 3d biped. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2849–2854. IEEE, 2004.

[5] Petar Kormushev, Barkan Ugurlu, Sylvain Calinon, Nikolaos G Tsagarakis, and Darwin G Caldwell. Bipedal walking energy minimization by reinforcement learning with evolving policy parameterization. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 318–324. IEEE, 2011.

[6] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338, 2016.

[7] Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel Van De Panne. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 36(4):41, 2017.

[8] Zhaoming Xie, Glen Berseth, Patrick Clary, Jonathan Hurst, and Michiel van de Panne. Feedback control for cassie with deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1241–1246. IEEE, 2018.

[9] Rolf Pfeifer and Josh Bongard. *How the body shapes the way we think: a new view of intelligence*. MIT press, 2006.

[10] Karl Sims. Evolving 3d morphology and behavior by competition. *Artificial life*, 1(4):353–372, 1994.

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

[12] Theo Jansen. Strandbeests. *Architectural Design*, 78(4):22–27, 2008.

[13] Shailen Agrawal, Shuo Shen, and Michiel van de Panne. Diverse motions and character shapes for simulated skills. *IEEE transactions on visualization and computer graphics*, 20(10):1345–1355, 2014.

[14] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[15] Charles Schaff, David Yunis, Ayan Chakrabarti, and Matthew R Walter. Jointly learning to construct and control agents using deep reinforcement learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9798–9805. IEEE, 2019.

[16] David Ha. Reinforcement learning for improving agent design. *arXiv preprint arXiv:1810.03779*, 2018.

[17] A toolkit for developing and comparing reinforcement learning algorithms. `https://gym.openai.com/docs/`. Accessed: 2010-09-30.

[18] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.