Learning Legged Locomotion with Deep Reinforcement Learning

Carlo Longhi
University of Bologna
`carlo.longhi@studio.unibo.it`

Abstract

In the past few years, Deep Reinforcement Learning techniques have become more and more successful in many robot control challenges. In this report is presented an implementation of the Proximal Policy Optimization algorithm for a legged robot locomotion task in a simulated environment. The results confirm the effectiveness of this approach, getting to learn useful gaits for two different robots.

1 Introduction

The design of a controller for the locomotion of legged robots has been, for a long time, a central research challenge. State-of-the-art approaches require an accurate model of the dynamics of the robot and the environment, often challenging to acquire. One of the most promising approaches to this task is Deep Reinforcement Learning (DRL). End-to-end DRL methods do not require any prior knowledge of the robot or the environment and, therefore, can completely automate the design of the controller.

Learning a controller directly on real robots is a challenging task for different reasons: robots are expensive, they break easily and require power. An even bigger challenge is how to acquire the large amount of samples required by DRL methods. Moreover, these algorithms are often sensitive to hyperparameters and need to be tuned accordingly, making things even more difficult. For these reasons, the learning process is carried out using a simulator: IsaacGym, Makoviychuk (2021). Simulators like IsaacGym offer a solution to the problem of sample acquisition, as they allow to run multiple environments in parallel, significantly reducing the amount of time required by this task.

The last component of the project is Revolve2 (https://github.com/ci-group/revolve2). Revolve2 is an open-source library that offers the possibility of simulating modular robots in IsaacGym. The robots are composed of three different types of modules:

- Core: the head of the robot, it contains the hardware required by the controller
- Brick: a simple squared block
- Active Hinge: the effector of the robot, made by an actuated hinge

The modularity of the robots and the simple structure of the modules make these kinds of robots very easy to build. The configuration of the hinges determine the state of the robot at each time step. Each action, computed by the robot controller, computes a new position for its hinges to take. Since the hinges position is represented by a continuous value in the range [-1, 1], the task is modeled as a continuing Reinforcement Learning problem with a continuous action space.

2 Methods

2.1 Proximal Policy Optimization

The challenge of learning the controller for the agents is addressed using a DRL algorithm called Proximal Policy Optimization (PPO), Schulman (2017). PPO is a scalable, data efficient and robust policy gradient method defined as follows.

Let π_θ, a stochastic policy and $\pi_{\theta_{old}}$ the stochastic policy before the update, then $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ denotes the probability ratio. So $r(\theta_{old}) = 1$, where θ_{old} is the vector of policy parameters before the update. The objective maximized in the PPO algorithm is defined as:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)]$$

where ϵ is a hyperparameter and $\hat{A}_t$ is an estimator of the advantage function at timestep t. The second term clips the probability ratio to keep $r_t(\theta)$ close to 1. The final objective, being the minimum between the clipped and unclipped terms, is a lower bound on the unclipped objective.

This implementation makes use of a Proximal Policy Optimization Algorithm (PPO) that uses fixed-length trajectories as follows: at each iteration, N parallel agents collect data for T timesteps and the surrogate loss is computed on these NT samples. Then, the surrogate loss is optimized with minibatch Stochastic Gradient Ascent.

2.2 Advantage Function

Advantage functions, by their definition $A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$, measure how much better is to take the action a in state s instead of following the current policy π. These types of functions, used to compute the policy gradient, are a useful tool to reduce variance. In practice, the advantage function is not known, and must be estimated. For this reason, the Generalized Advantage Estimator (GAE), Schulman (2015), computed using a learned state-value function $V(s)$, is adopted. In particular, this version of GAE uses a truncated version of the estimation, as in Schulman (2017):

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + ... + ... + (\gamma\delta)^{T-t+1}\delta_{t-1},$$

where, $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$.

2.3 Controller Architecture

The actions of the agent are chosen using a stochastic policy $\pi(\theta)$ where θ are the parameters of two neural networks: an actor network and a critic network. The choice to use separate networks has been found to lead to better performances, according to Andrychowicz (2021). The first network outputs a probability distribution over the action space, whether the second produces a single value representing the discounted future returns.

The two networks take the same inputs but experiments with different sets and combinations of observations have been carried out (see Section 3.2). Every type of observation is separately encoded by a linear layer in a 32-dimensional vector. The vectors obtained are then concatenated and encoded again in a 32-dimensional vector by another linear layer. Tanh non-linearities are used after each layer.

The resultant vector is then given in input to the final linear layer of both the networks. The continuous position of each actuated hinge is sampled from a Gaussian distribution. For this reason, the output of the final layer of the actor network is the vector of means for such distributions. In the critic network, instead, the output is a single value representing the state value. The standard deviations for the probability distributions are computed as the exponential of a parameter vector, initialized to [0,..,0].

2.4 Reward

For this task, a morphology-agnostic reward is used. This decision has been made to take full advantage of the modularity of the simulated robots, making it easier to test the same learning technique on multiple morphologies. The reward function is defined as:

$$r(t) = d(t) - d(t-1)$$

where $d(t)$ is the distance, from the agent's starting position, at timestep t. At each timestep, the agent receives a reward that is equal to the distance traveled away from the starting point. The reward, defined in this way, represents the speed at which the agent moves away from the starting position.

2.5 Loss Function

Since the architecture is composed of a pair of separate networks that do not share parameters, two separate loss functions have been used. For the actor network, the loss is defined by the PPO algorithm as L_t^{CLIP}, augmented by adding an entropy term to ensure exploration, as in Mnih (2016). The final loss is:

$$L_t^A(\theta) = \hat{\mathbb{E}}_t[L_t^{CLIP}(\theta) + cS[\pi_\theta](s_t)]$$

where c is a coefficient and S denotes an entropy bonus.

The loss function for the critic network is a squared-error loss

$$L_t^C(\theta) = \hat{\mathbb{E}}_t[-(V_\theta(s_t) - V_t^{targ})^2]$$

where V_t^{targ} is the return target, computed from the Advantage Function estimation as $V_t^{targ} = \hat{A}_t + V(s_t)$.

3 Experiments

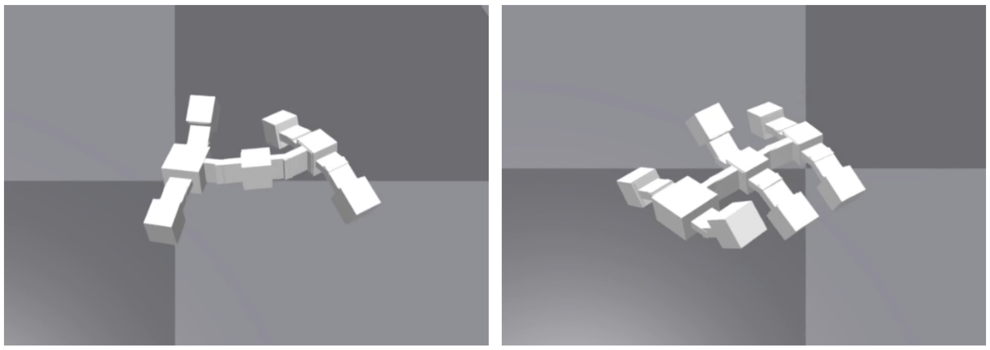

Figure 1: The two robots trained. The first, called "Gecko", has a body composed of 4 limbs with a total of 6 actuated hinges. The second robot, called "Ant", has 6 limbs and 8 actuated hinges

3.1 Settings

Two robots with different morphologies are trained using the methods described before, to learn locomotion on a flat plane. The robots, called Gecko and Ant, are shown in Figure 1.

Since the actuated hinges can take a value in the interval $[-1, 1]$, but the Gaussian distribution is defined over all $\mathbb{R}$, the sampled actions need to be clipped before being fed to the environment. All the experiments are carried out using the same set of hyperparameters: 200 iterations with 64 parallel agents, each iteration the agent performed 128 steps with 4 steps per second. The gradient ascent is performed using 4 epoch each iteration with batch size 2048, the actor and critic learning rates are respectively set to 0.0008 and 0.001. Adam, Diederik (2014), is used as optimizer. The learning rate is decayed linearly, as both Engstrom, Ilyas (2020) and Andrychowicz (2021) found it to be beneficial to get higher returns. For PPO, the clipping ratio ϵ is set equal to 0.2 and, for the advantage estimator, λ and γ are respectively set to 0.95 and 0.99. Finally, the weight for the entropy term in the actor loss is 0.01.

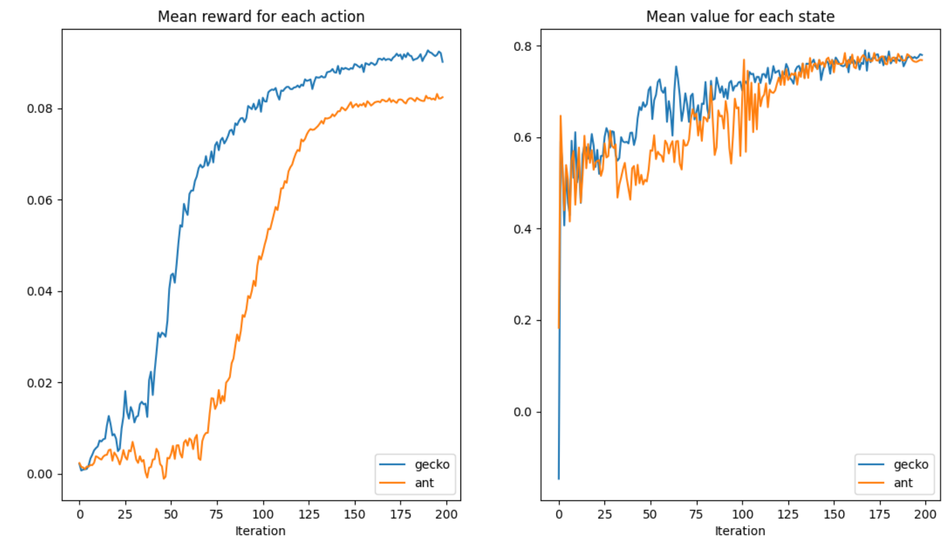

Figure 2: Results for the training of the two robots. The observations provided to the networks are the state of the hinges and the orientation of the core module

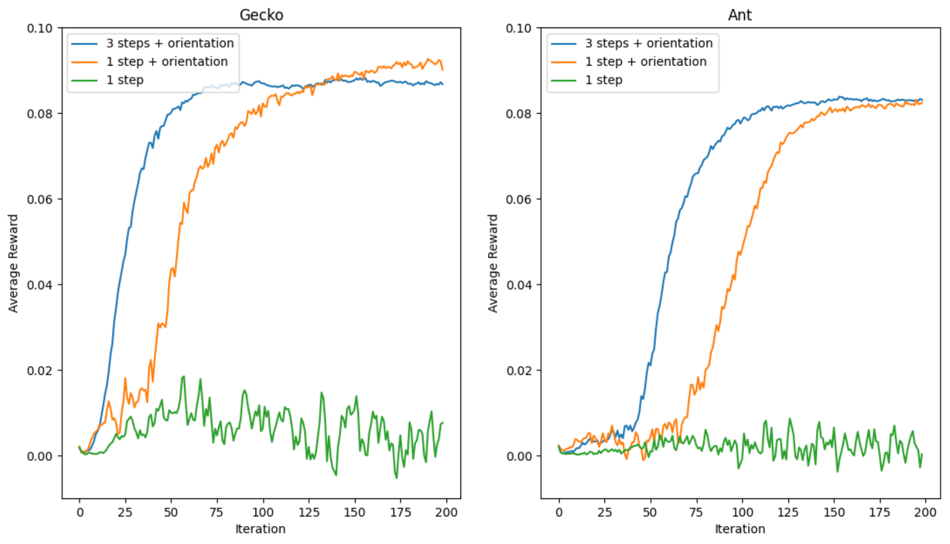

Figure 3: Results for the training of the two robots with different types of observations

3.2 Results

The first experiments were conducted providing in input to the networks, as observations, the position of the hinges and the orientation of the core module. The results of training the system with the two robots are reported in Figure 2. Both the Gecko and the Ant reach an average reward per action which is slightly above 0.08. The Ant, however, requires many more iterations to learn. On the other hand the reward for the Gecko increases starting from the first iterations, approaching its peak after only

75 of them. The Gecko also reaches higher reward values; this could be due to a morphology which is simpler and more suitable for legged locomotion. Indeed, when analysing the gait of the Ant, it is noticeable how it does not often use its central limbs, suggesting that they are not very useful. The average value for the states visited by the agents quickly improves, reaching its peak around the same time as the average reward, for both morphologies.

A second set of experiments is designed to test the effect of different types of observations on the training process. The results are reported in Figure 3. The experiments are carried out providing the controller with the last 3 positions of the hinges instead of just the current one. The idea behind this experiment is to test if the controller is able to learn some kind of relationship about the sequence of steps taken. As can be seen from the reward graphs, this type of observation proves to be very useful, speeding up the training of a useful gait.

Then, to assess the utility of the orientation information in the training process, a test is carried out using, as observation, only the position of the hinges. It is clear from the graphs that the orientation plays a central role while learning to maximize this reward. Indeed, both the robots are unable to perform any locomotion without receiving this information in the training process.

4 Conclusion

In this report is presented an implementation of a learning system for locomotion with legged robots. The robots testes, the Gecko and the Ant, both learn a useful gait, with the former achieving better results. The work shows an example of how can Deep Reinforcement Learning techniques be applied to many tasks in the field of robot control. The experiments performed prove the effectiveness of the Proximal Policy Optimization algorithm. They also suggest the possibility to extend this setting to more complex robotic control tasks by easily scaling up the size of the simulation, the complexity of the controllers, the number of observations or by using more challenging environments.

An issue with this approach is how well the controllers learned in simulation will adapt to real robots. Creating realistic simulations is a really hard problem and optimized controllers can take advantage of hidden errors in the software, making the learning process useless. Research that tries to tackle this problem will be crucial to effectively deploy robots in the real world.

References

Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, Gavriel State. (2021) "Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning" in arXiv preprint arXiv:2108.10470

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov. (2017) "Proximal Policy Optimization Algorithms" in arXiv preprint arXiv:1707.06347

J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. (2015) "High-dimensional continuous control using generalized advantage estimation" in arXiv preprint arXiv:1506.02438

Diederik P. Kingma, Jimmy Ba. (2014) "Adam: A Method for Stochastic Optimization" in arXiv preprint arXiv:1412.6980

Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, Aleksander Madry. (2020) "Implementation Matters in Deep Policy Gradients: A Case Study on PPO and TRPO" in arXiv preprint arXiv:2005.12729

Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphaël Marinier, Leonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, Olivier Bachem. (2021) "What Matters for On-Policy Deep Actor-Critic Methods? A Large-Scale Study" in International Conference for Learning Presentations (ICLR)

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, Koray Kavukcuoglu. (2016) "Asynchronous Methods for Deep Reinforcement Learning" in arXiv preprint arXiv:1602.01783