Reinforcement Learning for Legged Locomotion

Yathartha Tuladhar and Taylor Apgar
Oregon State University
Corvallis, Oregon
tuladhay, apgart, @oregonstate.edu

Abstract

Hand engineering complex behaviors on high degree-of-freedom (DOF) robots can be very hard or intractable. One of the reasons for this is that it can be very hard to model the robot dynamics and to write rules for a desired behavior. Deep Reinforcement learning offers a set of tools for the design of rich, sophisticated, and hard-to-engineer behaviors. ***In this project, I demonstrate how a reinforcement learning algorithm with a neural network policy, can be used to teach a bipedal robot to stand*** *in a realistic physics simulator. For reinforcement learning,* ***I explore two action spaces, namely operational space and the torque space.*** *In both cases, the neural network policy starts off knowing nothing about how to stand, and at the end learns to stand naturally.*

For result videos, please refer to the final presentation submission.

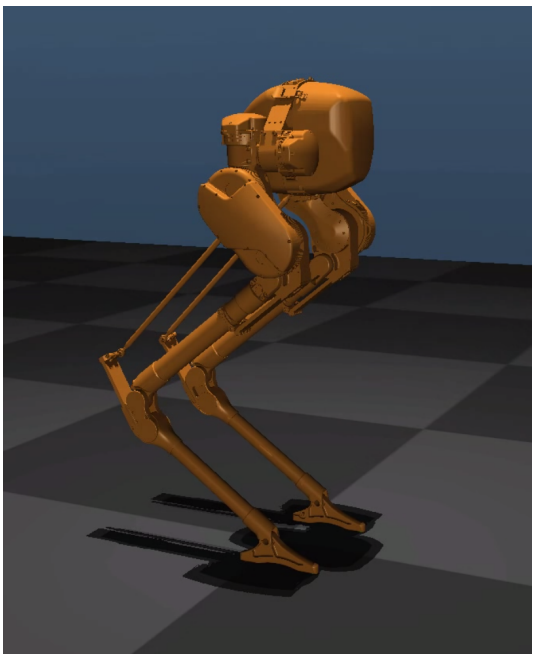

Figure 1. Robot Cassie standing using learned MLP policy trained with Trust Region Policy Optimization (TRPO)

1. Introduction

Traditionally most controllers for robots are hand engineered, i.e., the control engineers have to write down a set of rules (based on the dynamics of the robot) that tell the robot what to do based on the internal states of the robot and sensing information from it's environment. This strategy can work very well for lower DOF systems, and where the behaviors exhibited are not very complex. However, hand engineering rules for high DOF systems can be very hard, or **in most cases it is not possible to write down a set of rules or steps that the robot/agent should take while doing a complex task** such as doing a back-flip (on a legged robot) or drifting (on a fairly fast mobile robot/car). Moreover, writing down a set of rules or steps require that one has a very good model of the robot and how it interacts with the environment. Creating an accurate model can be very hard or intractable for high DOF systems. For example, a bipedal robot has a highly-nonlinear dynamics. Moreover, it's dynamics change based on phase of the gait that it is in. That is, stance phase (with one foot on the ground) will have a different set of dynamics than flight phase (when both feet are off the ground). Thus it is hard to model the system as well as it's interaction with the environment. Current approaches for bipedal robots, namely Zero Moment Point, trajectory optimization, and Model Predictive Control need an accurate model of the robot and the environment.

Deep Reinforcement Learning can be used as a general purpose framework for designing controllers for non-linear systems. Traditionally, reinforcement learning was applied to problems with low dimensional state space, but use of Deep Neural Networks as function approximators with reinforcement learning have shown impressive results for control of high dimensional systems [4]. Deep reinforcement learning has demonstrated remarkable progress in learning tasks such as playing Atari games, 3D naviga-

tion tasks and board games. It enables a robot (an agent) to autonomously discover desired behaviors through trial-and-error. This motivates the use of learning a policy (controller) that is able to learn how to interact with it's environment to perform a certain task by trial and error. Recently researchers have shown that robust locomotion behaviors can emerge from simple policies when trained in a rich environment [1].

The goal of this project was to explore how Reinforcement Learning can be used to train a high DOF robot to stand. In this project I setup a simulation environment in a physics simulator (MuJoCo) for the robot to train, and use the Trust Region Policy Optimization (TRPO) from RL-LAB [6], which is a reinforcement learning library. **Figure 1 shows the robot Cassie standing with the policy learned in operational space.**

2. Background

2.1. Artificial Neural Networks

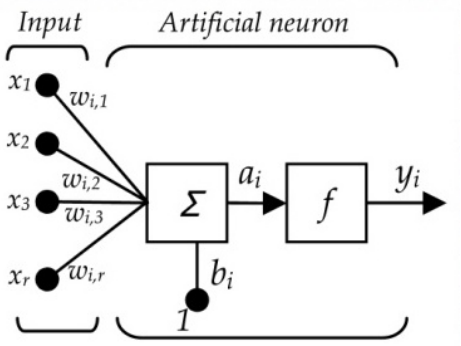

Figure 2. An artificial neuron

An artificial neural network is a set of interconnected neurons, arranged in layers, where each neuron takes some real valued inputs and gives out a real valued output. The "neurons" in this sense are merely inputs and outputs to a mathematical operation. An artificial neuron is shown in figure 2. The network takes an input in the form of vector *x*, and calculates the weighted sum of inputs by multiplying with it's respective weights *w*. A bias *b* is added to this weighted sum, which results in *a*. This *a* is then passed through an activation function *f*, to produce the output of the neuron *y*. Thus, the equation for an individual neuron can be written as $y_1 = f(\sum_j x_j w_{ij} + b_i)$, where *j* is the number of inputs to the neuron. The activation function introduces nonlinearity into the output, which helps the network learn nonlinear representations from training data.

A feedforward network is where the neurons are arranged in layers, and each layer can have different number of neurons. The layers are usually categorized as *input, hidden* and *output* layers. The information flow from *input* layer to *hidden* layers, and finally to *output* layer, where the output is generated. The *weights*, and *biases* in each layer can be trained using a technique known as *backpropagation*.

Artificial Neural Networks can be used to learn a wide variety of nonlinear functions. It has been widely used in robotics by researchers, in a supervised learning setting to teach robots certain tasks. For this project we will be using a feed-forward neural network to learn the task of standing on the robot Cassie.

2.2. Deep Reinforcement Learning

Deep Reinforcement Learning refers to the use of deep neural networks as function approximators in a reinforcement learning framework. Deep reinforcement learning has shown promising results in ATARI games [2] and flying helicopters, where it has achieved human level performance. Deep reinforcement learning can handle high dimensional inputs and outputs, which make it suitable for the purpose of using learning on a bipedal robot.

The main components for implementing deep reinforcement learning are: **states, policy and actions**. The states are the current states of the robot, or some observation from the environment which the policy can use to compute an appropriate action. The policy evaluates the states and produces outputs, which are referred as actions. When this action is executed on the environment, a reward is generated, and the states are updated. The goal of reinforcement learning is to learn a policy that maximizes the expected sum of all the future rewards.

2.3. Policy Improvement

Initially the neural network policy is initialized with random weight parameters. Policy gradient methods can be used for improving the policy [5]. In these methods, a certain number of trajectories(rollouts) are executed using the current policy, and rewards are collected along the way. After a number of rollouts have been executed, the transition data (states, actions and rewards) for each trajectory can be used to update the policy. The vanilla policy gradient algorithm, REINFORCE is shown in figure 3.

REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run it on the robot)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)\left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Figure 3. REINFORCE Algorithm

A slightly more complex algorithm that uses the same policy gradient method is the Trust Region Policy Optimization (TRPO). TRPO incorporates a constrained optimization procedure that ensures monotonic improvement to the policy. It does this by limiting the KL divergence of the

new policy with respect to the old policy during an update. The algorithm for TRPO is shown in figure 4

for iteration=$1, 2, \ldots$ **do**
Run policy for T timesteps or N trajectories
Estimate advantage function at all timesteps

$$\underset{\theta}{\text{maximize}} \sum_{n=1}^{N} \frac{\pi_\theta(a_n \mid s_n)}{\pi_{\theta_{\text{old}}}(a_n \mid s_n)} \hat{A}_n$$

$$\text{subject to} \quad \overline{\text{KL}}_{\pi_{\theta_{\text{old}}}}(\pi_\theta) \leq \delta$$

end for

Figure 4. TRPO Algorithm

3. Problem Setup

For this project I use Multi Joint Dynamics with Contact (MuJoCo) as the simulator. A realistic model with the actual physical properties of the robot Cassie was used for simulations. The simulated robot was constrained to two dimensions, making it move only in the x and z directions. It could also rotate in plane but cannot rotate in or out of the plane. Using this simulator I was able to sample trajectories from the policy. Figure 5 shows the state, policy and actions for my problem setup for this project.

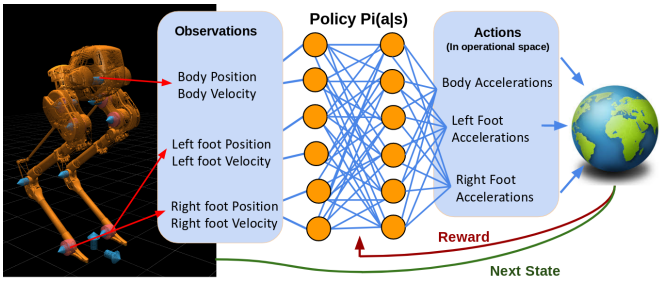

Figure 5. States, policy, and actions for the Reinforcement Learning Paradigm

An environment that was compatible with the RLLAB[6] framework was developed for the Cassie robot. This gave me access to using reinforcement learning algorithms in the library. The ***states*** (observations) are the x, z, rotation (in plane) positions and velocities of the center of mass of the body, the x, z, rotation velocities and the relative x, z and rotation positions of the two feet with respect to the body. Thus in total, there were 18 inputs as shown in table 1

The ***policy*** was chosen to be a Gaussian multi-layer perceptron (MLP) with two hidden layers. Each hidden layers had 32 units. There were 18 inputs to the MLP corresponding to the 18 input states. The output of the Gaussian MLP is an action sampled from a Gaussian with some mean and variance.

State	Size
Body Position	3
Body Velocity	3
Left Foot Position	3
Left Foot Velocity	3
Right Foot Position	3
Right Foot Velocity	3
Total	18

Table 1. States/Observations for Reinforcement Learning

Two ***action spaces*** that were explored in this project are: the operational space, and the torque space. [3] compared the effect of action spaces in reinforcement learning for simulated robot models. They found that the choice of action space does make a difference in the learned policy. For this project, the operational space the actions are the x and z accelerations for the end effectors (i.e., the left foot, right foot, and the center of mass) of the robot, plus the acceleration for the body rotation (to control the pelvis pitch if needed). Thus the size of the actions was an array of 7. An underlying controller would turn these desired accelerations into torques with some the proportional-derivative (PD) gains. In this way the operation space actions have a layer of abstraction. In torque space, the actions were the raw torques for the individual motors of the robot. Thus in total there were 6 actions corresponding to the 6 motors (3 in each leg) of the robot.

Reward shaping is an important aspect of reinforcement learning, since it drives what the policy will learn to maximize. For the purpose of standing, the reward function can be shaped in multiple ways. The set of reward function that worked best for making the robot stand is shown below:

while *not at end of* ***n*** *timesteps* **do**
initialize reward to 0;
reward = 0;
reward -= 5*$(targetHeight - bodyHeight) * *2$;
reward -= 2 * $(leftFootPosition) * *2$;
reward -= 2 * $(rightFootPosition) * *2$;
reward -= 0.01 * $sum(action * *2)$;
reward +=1;
end

The reward is initialized at 0. The first reward encourages actions that achieve the desired target height for the robot. The second and third reward encourages actions that put the foot closer to CoM in the x direction. The fourth reward encourages to use less actions (accelerations/torques based on the action space). The final reward helps in the learning process by encouraging actions that keep the robot alive every time step. The robot **resets**, when the body

height falls below 0.5 meters.

The best performing hyperparameters for TRPO algorithm are as follows:

Initial Standard Deviation = 2.0
Batch Size = 15000
Max path length = 1000
Iterations = 5000
Discount factor = 0.99
Step size = 0.005

It is important to note that the control loop was running at a rate of 200 Hz. Thus the max path length of 1000 would correspond to a maximum time of 5 seconds per trajectory.

4. Results

Figure 1 shows the robot Cassie standing with the policy learned in operational space.

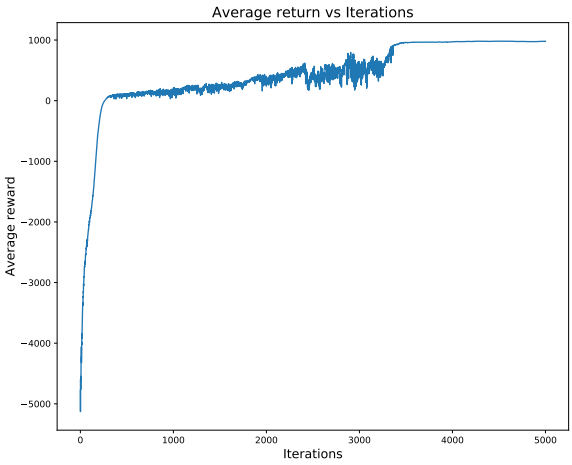

Figure 6. Average return vs iterations (for operational space)

The Gaussian MLP policy was able to learn how to make the robot stand using both action space, i.e., the operational space mode, and the torque mode. For both the action spaces, the reward shaping is the same as the one described in previous section. Figure 6 shows the average return versus iterations for learning the policy in operational space. Figure 7 shows the standard deviation of return vs. the number of iterations. Figure8 shows the standard deviation of the Gaussian MLP policy vs iterations.

Figure 9 shows the average return vs iterations for learning the policy in torque space. Figure 10 shows the standard deviation of return vs. iterations. And figure 11 shows the standard deviation of the Gaussian MLP policy vs iterations.

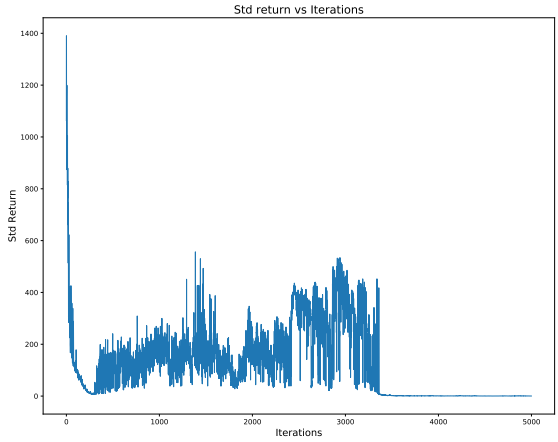

Figure 7. Std return vs iterations (for operational space)

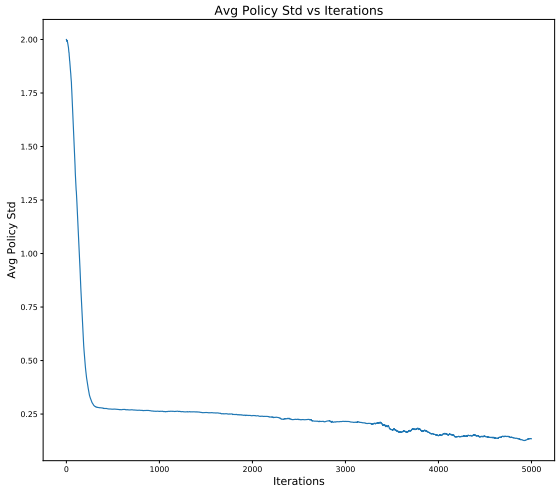

Figure 8. Average Policy Std vs iterations (for operational space)

5. Discussion

Finding a good set hyperparameters was not very intuitive. Moreover I did not find any literature that explained a way to tune or choose the hyperparameters. While choosing the batch size and the max path length, one needs to be careful such that even when the rollout of the policy does not reset (due to falling a certain height in this case), it should be able to run about 10 trajectories just so that it is still exploring different actions.

The final learned policy was very sensitive to the reward shaping. In my experience, choosing rewards that result in less variance leads to better learning. For example, in order to learn how to not fall, adding a small positive reward for every time step that it was alive was better than having a

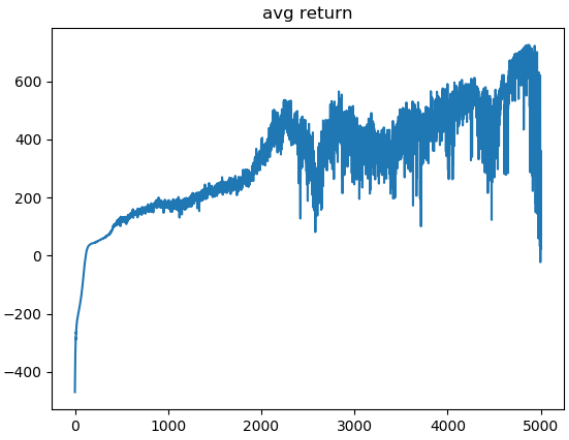

Figure 9. Average Return vs iterations (for torque space)

Figure 10. Std return vs iterations (for torque space)

huge negative reward when it reset (by falling below a certain height).

In the future, I would like to explore model based reinforcement learning approaches in order to make the robot do more complex things such as walk, run, and jump.

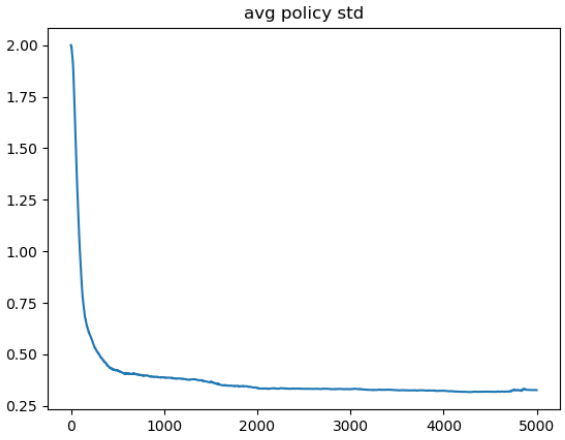

Figure 11. Policy Std vs iterations (for torque space)

References

[1] Nicolas Heess, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, Ali Eslami, Martin Riedmiller, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[3] Xue Bin Peng and Michiel van de Panne. Learning locomotion skills using deeprl: does the choice of action space matter? In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, page 12. ACM, 2017.

[4] Divyam Rastogi. Deep reinforcement learning for bipedal robots. 2017.

[5] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*, pages 5–32. Springer, 1992.

[6] Rein Houthooft John Schulman Pieter Abbeel Yan Duan, Xi Chen. Benchmarking deep reinforcement learning for continuous control. *International Conference on Machine Learning (ICML)*, 2016.