Comp 138 RL: Homework 3

Matt McDermott

November 2, 2020

Background

Thus far, the problems encountered in this course have had both discrete action and output spaces. While approximating continuous systems into discrete states may work for some situations, highly nonlinear systems such as those encountered when working with multibody dynamics can not be easily approximated with a discrete simulation. The application of using reinforcement learning on a simple game may seem insignificant, however, the techniques used to arrive at a solution for controlling the joints of a 2D stick figure without having access to the underlying dynamics can be applicable to many real world continuous control problems.

Problem Statement

The goal of this project was to create a policy capable of controlling the joint torques of a 2D humanoid character. This project was inspired by the early 2000s flash game *QWOP* in which a human player must individually control the torque outputs of a runner's four knee and hip joints. *QWOP* is incredibly difficult and a player's score is determined not by how fast they can run but by how far they are able to move the character down the track before falling over.

Methods

The project began by first recreating the *QWOP* environment using the popular pymunk 2D physics engine. I originally considered simply modifying the 2D_Walker environment from OpenAI's gym, however, I found that remaking the environment from scratch gave me more freedom to mess with some of the lower level parameters of the rigid body physics in order to make the character behave similar to the original game.

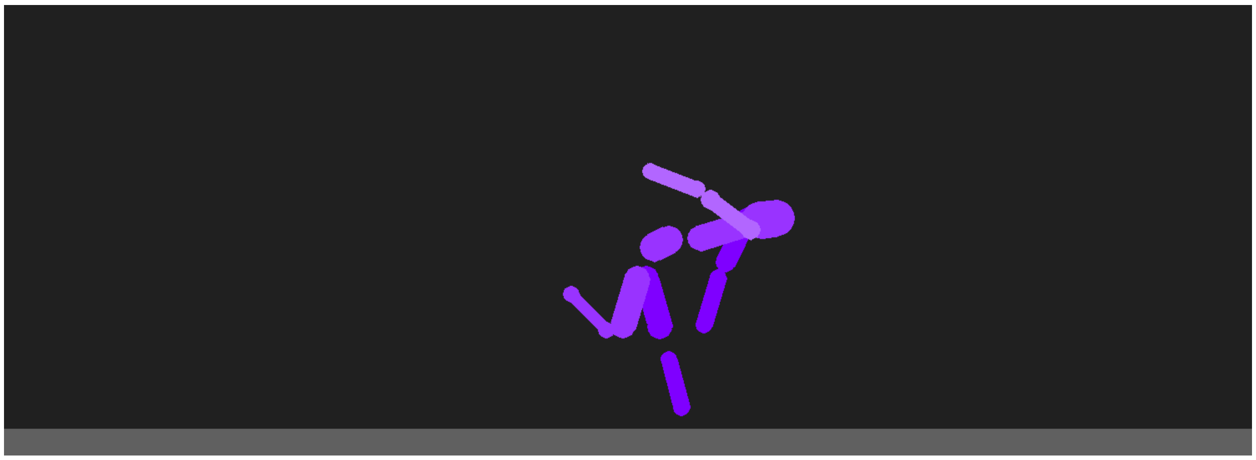
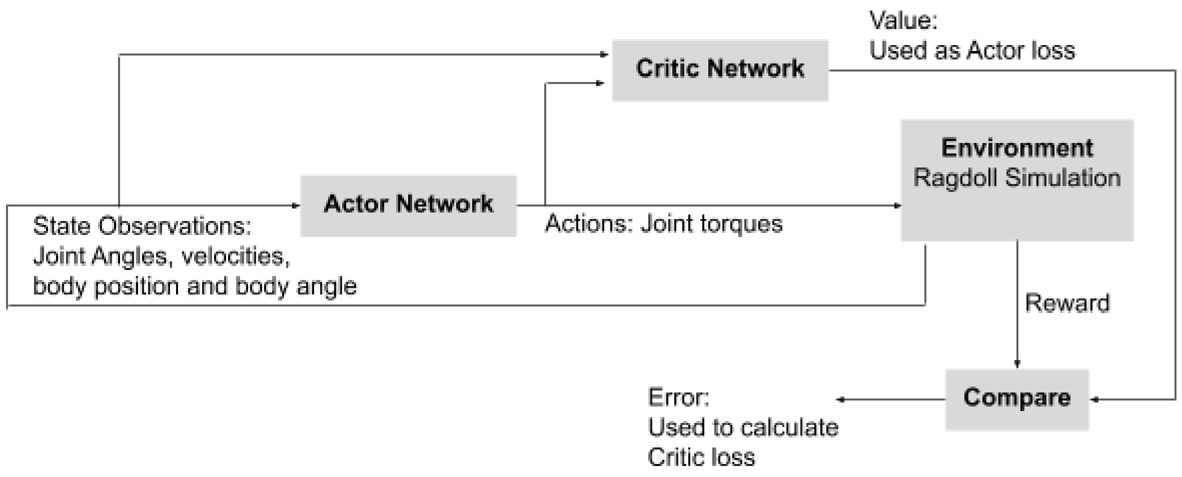


Fig. 1: Player being tossed around in the simpleHuman demo environment

I created a script in which the left and right knees, hips and back of the character can be controlled with the Q, W, O, P, and E keys respectively. This was used to tune parameters such as ground friction, mass of each link and overall viscosity of the system. Rather than adding ankle joints, the behavior of the foot was modeled as a simple spring at the end of each lower leg segment. My original intention was to include the joint states of the character's arms as observable but uncontrollable states, such that the overall character would be acting as an underactuated system (a particularly difficult classical control problem), however, the training times even without arms were long enough to scrap that idea for the time being.

As a benchmark, I first attempted to solve this problem with the same tabular Q-Learning methods used on the previous "drifting" exercise. Even for what may be considered the "simple" 2D case of a walking humanoid, modeling ground contact makes it incredibly difficult to obtain a closed form model of the system. Without a model of the environment, it is not possible to determine state transitions, so Q values must be estimated from state-action pairs. In the most simple case there are 13 states (5 joint angles, 5 joint velocities, height of body off of ground, angle of body, angular velocity of body) and 5 actions (torques to apply at each joint). Even with discretizing all values of torque to either -1, 0, or 1 units and discretizing the 13 states to 5 values (which is horribly inaccurate), it produces $5^{13} * 3^{5} = 2.9e10$ entries for a Q table. Just initializing an array this large takes up more than 29GB of RAM, which is significantly more memory than my machine can provide.

This problem was solved with an Actor-Critic based Deep Deterministic Policy Gradient (DDPG) implementation. The DDPG approach uses two separate neural networks to solve the issues presented by continuous action and output spaces. The first network is the Actor Network and is constructed similarly to a network used in a classification task. The Actor Network takes in continuous observations of the states of the agent (joint angles, velocities, etc.) for each node of its input layer and outputs torque values for each respective joint on each node of the output layer. Because the output of the Actor Network is continuous however, it is impossible to generate a Q-table from these values to keep track of how successful the resulting actions are based on the current policy. For that reason, a separate Critic Network is introduced that takes in state-action combinations and outputs a single Q value to describe how the Actor performed.



At the beginning of training, both networks are initialized with random weights. A replay buffer was created to store a Tensor of [s, a, r, s’, d] for each step, with the last entry *d* being a boolean that described whether the entry was the last entry of a trial (meaning that it either led to falling on the ground or was the last action taken before the time limit was reached). Batches of size 10 were taken from this buffer and were sampled randomly to train both agents. As explained by Yoon’s article, if both networks are trained simultaneously, training time will suffer significantly due to the fact that the Q values on both sides of the critic training equation will be updated simultaneously. For that reason, Target Networks for both the actor and critic are generated using a time delayed copy of each other's respective policy. This means that the critic network can be effectively “frozen” while the actor network is trained and vice versa.

Both the actor and the critic networks contained two hidden layers. Because the joint torques are both positive and negative, the tanh() function was applied to squeeze the output layer of the actor network between -1 and 1. Exploration was encouraged through the addition of Ornstein-Uhlenbeck noise in the action space. This meant that the torque commands sent to each joint were modified by a normally distributed random walk rather than pure Gaussian noise. In my trials I had success without decaying this parameter over time, though I did find that OU noise outperformed pure Gaussian noise.

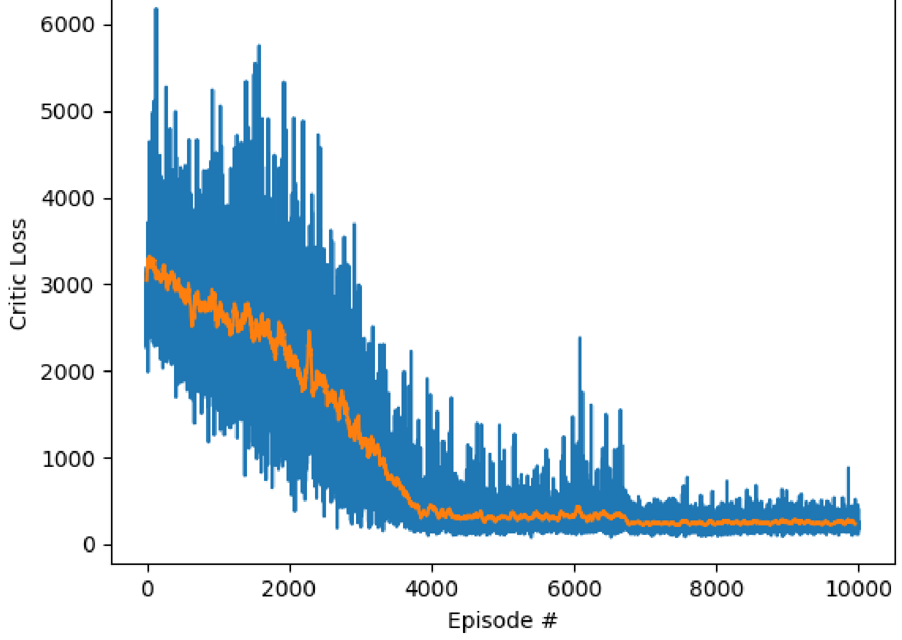
The reward function used in this simulation was:

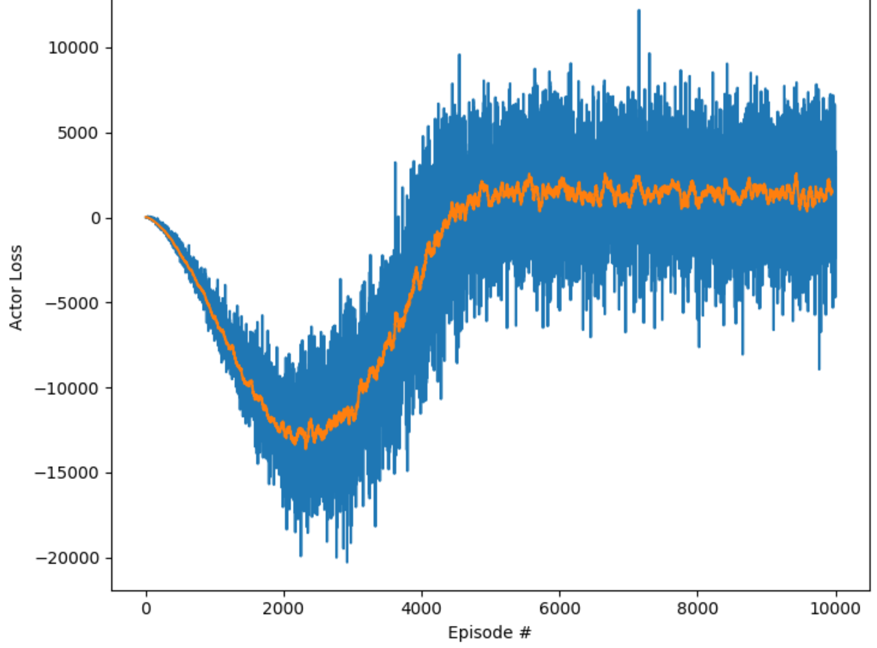
$$r = hip_x - hip_y^{\,2} - 100(sum(torqueVector^2)) - fallPenalty$$

Where:

hip_x	=	x position of the hip (in pixels)
hip_y	=	y position of the hip (in pixels, positive is down)
torqueVector	=	vector of toques sent to joints (-1 to 1)
fallPenalty	=	penalty factor set to 10,000 if the player falls over

Results


Critic Loss
Episode #
0
1000
2000
3000
4000
5000
6000
0
2000
4000
6000
8000
10000


Actor Loss
Episode #
10000
5000
0
−5000
−10000
−15000
−20000
0
2000
4000
6000
8000
10000

Analysis

It is amazing how good the agent was at finding ways to exploit the reward system without actually learning to walk. One run involved the character performing a series of frontflips across the platform, which was later fixed by triggering the *fallen* flag when the agent's torso exceeded more than +/- 2rad from vertical.

According to the OpenAI gym leaderboard, the best algorithms manage to train the simplified 2D-Walker environment in 300-500 trials. Given that my environment had an extra degree of freedom (bending at the waist) it makes sense that the QWOP agent took longer, reaching convergence around 4,000 trials. It is likely that I had included more neurons for each of the hidden layers than necessary. If I were to continue this project I would likely attempt to use some form of optimization algorithm to find the ideal side of the hidden layers.

Conclusion

Based on my experience with this project, DDPG is not nearly as robust of a method as some of the tabular based strategies explored previously. Given a problem that can be linearized, a Q-learning method would likely be much easier to implement. Besides designing the network and adjusting hyperparameters which were non-trivial but expected, I spent a significant amount of time adjusting the reward function to encourage walking behavior. That being said, it is an incredibly useful tool for approaching problems that would be nearly impossible with classical control methods.

References

Sutton, Richard; Barto, Andrew (1998), *Reinforcement Learning*, MIT Press, ISBN 978-0-262-19398-6, archived from the original on 2013-12-11.

Yoon, C., 2020. *Deep Deterministic Policy Gradients Explained*. [online] Medium. Available at: <https://towardsdatascience.com/deep-deterministic-policy-gradients-explained-2d94655a9b7b> [Accessed 3 November 2020].

Fussell, L., 2018. Exploring Bipedal Walking Through Machine Learning Techniques. *University of Edinburgh Press*, [online] Available at: <https://project-archive.inf.ed.ac.uk/ug4/20181201/ug4_proj.pdf> [Accessed 3 November 2020].

http://www.foddy.net/Athletics.html