Eloka Obi

Deep Reinforcement Learning From Scratch: Final Project

Introduction

The goal of our project was to use deep reinforcement learning towards a locomotion task. Specifically, we explore Deep Q-Network (DQN) and Twin Delayed Deep Deterministic Policy Gradients (TD3) on the Swimmer simulation environment and problem statement provided by MuJoCo, through OpenAI Gym. The goal in this problem is to develop a control policy to make the simulated multibody swimmer move to the right (positive X-direction).

The swimmer is a 3-link multibody, with two rotors that connect each pair of links.

The 8-vector state is specified by the swimmer's heading angle and heading angle rate, the rate of change in the x and y coordinates of the swimmer's head, and the angle and angle rates of both rotors.

Actions are specified by a pair of rotor torque commands, (torque1, torque2), each between -1Nm and 1Nm.

The reward is a combination of the speed of x-coordinate of the swimmer's head and a penalty on the squared sum of the torque commands.

Deep Q-Network (DQN)

The DQN algorithm trains a neural network to approximate the function that maps state-action pairs to expected returns. Then, an epsilon-greedy policy determines which actions are followed at any given starting state, by maximizing the . To achieve this function approximation, the algorithm keeps a revolving history of experienced state-action-new state-rewards samples, then randomly selects from this history as training data points to break continuity in the training data. The neural network is then piecewise optimized with the objective of getting closer to the effect of the canonical bellman update.

Our implementation borrowed from the previous homework implementation on the cart-pole problem. We discretize the continuous torque space into steps of size 0.1Nm. This effectively gave 21x21 torque pairs to structure the output layer of the neural network.

Twin Delayed Deep Deterministic Policy Gradients (TD3)

The TD3 algorithm is an off-policy algorithm, a successor to DDPG aimed to improve its hyperparameter stability, one of the most used algorithms for continuous control problems in robotics and autonomous driving. It was introduced in Addressing Function Approximation Error in Actor-Critic Methods, in 2018. It uses actor-critic networks, common in reinforcement learning for continuous action spaces - the actor weights are used as the actions. TD3 has two parallel critic networks, which both estimate the Q-function according to the Bellman equation. At each iteration the algorithm picks the lowest of the two critics, subject to added noise regularization to encourage exploration.

The Q-value critics each have their own parameters that take observation S and Action A as inputs and return the corresponding expectation of the long-term reward. The algorithm also implements identical networks for the actor and critic to provide the targets for training, which improves stability when less-frequently updating the target and actor networks' parameters.

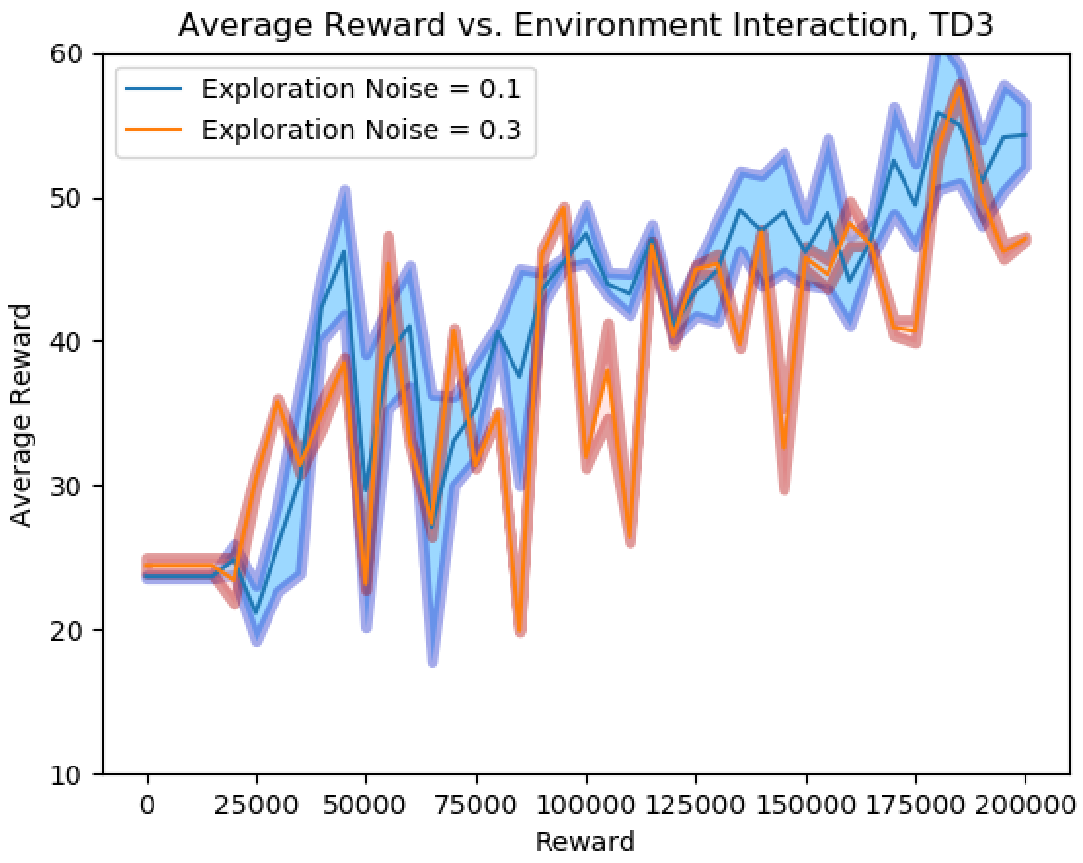
Conclusions

DQN

For a set of 300,000 environment interactions, the implementation of DQN took on the order of 12hrs without using GPUs. The results showed a quick adjustment away from negative rewards and then stabilized around 35 per episode. We experimented with two different final epsilons, for which there were no qualitative differences. To further develop this implementation, we would extend the training times to many more episodes, perhaps a 1000. Another possibility for improvement would be to encourage more exploration by increasing the minimum epsilon and decreasing its decay rate. The large output layer may need more varied samples to capture more dynamics of the environment.

TD3

For a set 200,000 environment interactions, the implemented TD3 algorithm took approximately 90 minutes to run using a GPU on Google Colab. To benchmark our implementation, we compare our performance to OpenAI's Swimmer Tensorflow Benchmark as well from the Python package stable_baselines. Running the stable_baselines benchmark, the TD3 algorithm was able to achieve a peak reward of 38.5 over 500,000 environment interactions. From OpenAI's benchmarks, they were able to achieve rewards in the range of 70 with 2+ million interactions. Our graph of rewards vs. iterations per time is comparable to both, achieving rewards that exceed 50.



The TD3 algorithm is a well-studied and well-implemented algorithm, so although the implementation required significant debugging there were many resources (cited in our code) that were able to help us out. Additionally, the original paper includes all of their hyperparameters, which we use as a default. We then investigated the result of tuning the *exploration noise*, that is, the mean of Gaussian noise that is added to the action in order to encourage exploration. We can see that by changing the exploration noise from 0.1 (the default) to 0.3, our rewards per epoch become more noisy, with both higher maxima and lower minima. This is expected as additional exploration may lead to better policies, though with less stable performance during training.

In future work, we would like to further investigate the effectiveness of hyperparameter tuning, as well as testing the algorithms on more complicated locomotion tasks. The original intention was to implement the same algorithms with OpenAI's MuJoCo Ant, though the Swimmer was chosen as computationally is much less expensive (observation space of 8 vs. 111).