Continuous Control Report

By Brandon Suen

Approach

For this project, I implemented the Deep Deterministic Policy Gradient (DDPG) algorithm using neural networks built using PyTorch. DDPG involves an actor network that selects an action deterministically and a critic network that represents the action-value function. This actor-critic format makes DDPG great for continuous action spaces, which is why it's a good fit for this project. I started off with a basic implementation of the algorithm with an experience replay buffer, local actor/critic networks that copy their weights to target actor/critic networks, and noise generation with the Ornstein-Uhlenbeck process. I updated the action selection to use an epsilon-greedy-esque strategy. There's a one minus epsilon chance that the action is chosen completely at random, and if the action isn't chosen at random, the amount of noise generated is multiplied by epsilon. I also introduced a noise multiplier to balance out the effects of multiplying by epsilon. This epsilon strategy allowed for more exploration in the beginning and more accuracy later in training, which yielded significant improvements. Despite these changes, I kept the original actor and critic models, which worked well. The actor network has 3 linear layers. ReLU activation functions are applied to the first two layers, and a hyperbolic tangent function is applied to the third. The critic network has a similar linear layer structure with a slightly modified number of nodes. A ReLU activation function is also applied to the first 2 linear layers, but no activation function is used on the third layer. Also, the result of the first ReLU activation function is concatenated with the actions taken.

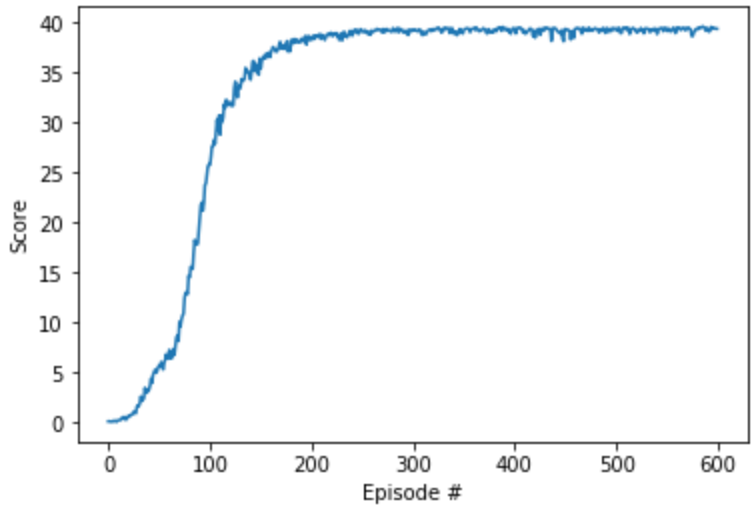
Hyperparameter Tuning

The first hyperparameter I changed was the epsilon decay rate. I had to set it high enough so that the agent did enough exploration, but low enough so that training didn't take an unnecessarily long amount of time. I landed on an epsilon decay rate of 0.99. Raising the update period to 20 and lowering the critic network learning rate to 1e-4 also resulted in significant improvements. The final hyperparameter values I ended up with are:

- Actor network learning rate: 1e-4
- Batch size: 128
- Buffer size: 1e5
- Critic network learning rate: 1e-4
- Critic network weight decay: 0
- Epsilon decay rate: 0.99
- Epsilon ending value: 0.01
- Epsilon starting value: 1
- Gamma: 0.99
- Noise multiplier: 2
- Tau (interpolation parameter when copying weights to target model): 1e-3

- Update period: 20

Results



The agent was able to meet the required mark of averaging a score of 30 over the last 100 episodes around episode 170, and it got up to an average of 39.32 over the last 100 episodes by episode 600.

Future Ideas

There are many improvements I could make to my implementation. For example, I could use techniques from the D4PG algorithm that's more optimized for multi-agent environments.