Tennis Report

By Brandon Suen

Approach

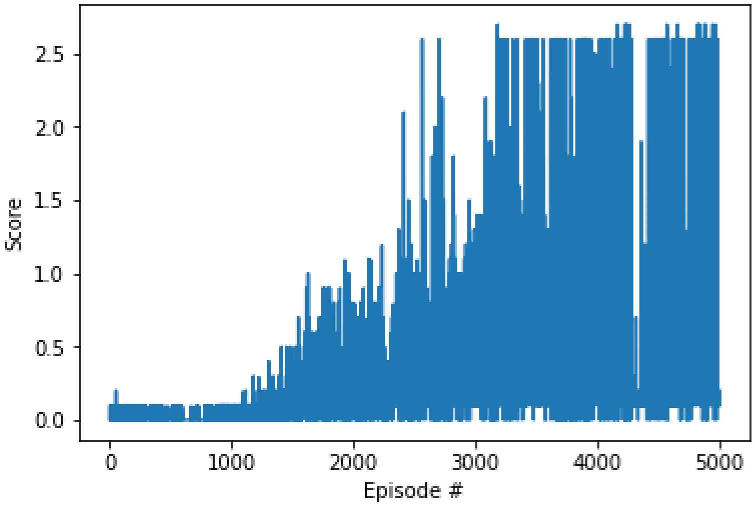
For this project, I implemented the Deep Deterministic Policy Gradient (DDPG) algorithm using neural networks built using PyTorch. DDPG involves an actor network that selects an action deterministically and a critic network that represents the action-value function. This actor-critic format makes DDPG 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 that I used from the Continuous Control project 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 the epsilon-greedy-esque strategy that I applied to the Continuous Control project. 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.

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 had an epsilon decay rate of 0.99 for the Continuous Control project, but this environment required a much higher decay rate because of how short the episodes are in the beginning of training and how little is learned from them. I landed on an epsilon decay rate of 0.9992. I lowered the update period from 20 to 5 and raised the critic network learning rate from 1e-4 to 5e-4, which 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: 5e-4
- Critic network weight decay: 0
- Epsilon decay rate: 0.9992
- 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: 5

Results



The agent was able to meet the required mark of averaging a score of 0.5 over the last 100 episodes around episode 3,200, and it got up to an average of 1.09 over the last 100 episodes by episode 4300.

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.