Notice

This report uses the same algorithm as my continuous control project for Udacity. As such, there will be very few differences between the two reports. Please navigate to the end of each section of the report for the content that has changed (with regards to the program learning over the actions of the two agents sharing the same model and agent object).

Learning Algorithm Overview

The learning algorithm present in the program is Deep Deterministic Policy Gradient, or DDPG, which comes from and improves upon Deep Q-Learning. The algorithm makes use of two sets of neural networks, one deciding the expected reward from taking certain actions, and one which proposes which actions to use. They both iterate off of the same incentive (that being the rewards from the environment). These networks are shared between the two AI's whom use it to learn at a faster and more elaborate pace than attempting to train two separate AI's.

Learning Algorithm In-Depth

DDPG ends up being a more valuable tool than Deep Q-Learning because of its capabilities of predicting rewards as well as its prediction of best actions. Though Deep Q-Learning is capable of predicting its own actions, DDPG tunes and recalculates the possible reward for an action mostly without seeing the state before in its lifetime. This makes for a much more robust system that can determine the best rewards and the best actions *independently* from each other. DDPG also works off of the same mechanics as Deep Q-Learning such as replay buffers and soft updates, which both help to dissociate states from other states.

Hyperparameters

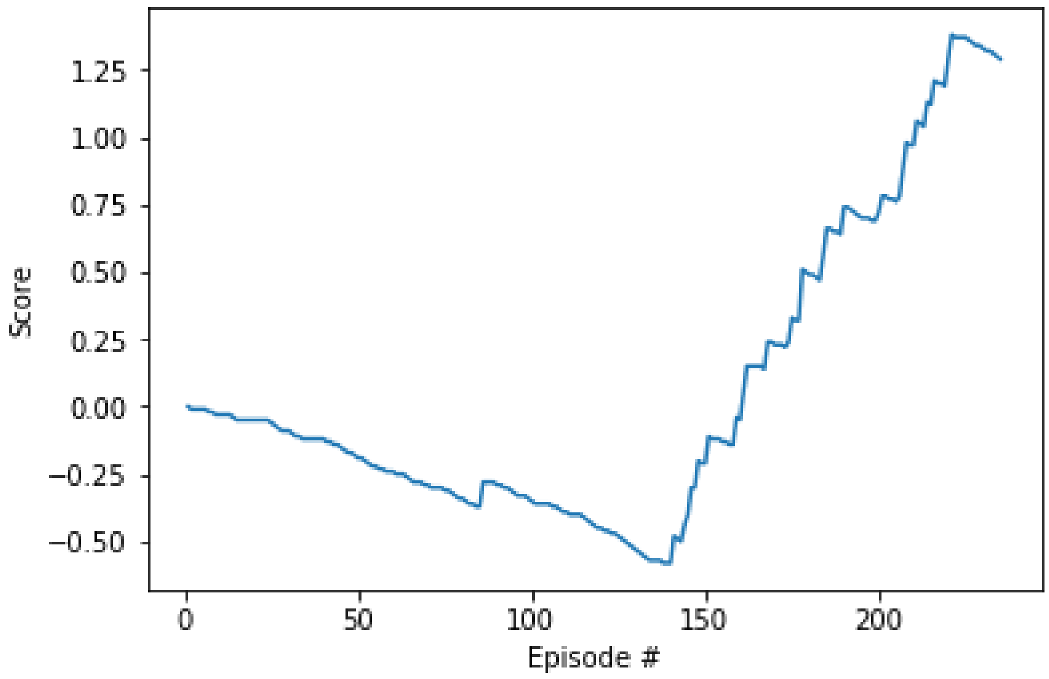
The chosen hyperparameters are a buffer size of 10^5, batch size of 16, gamma of .99999, tau of 10^-3, learning rate of 10^-4 (for the actor), learning rate of 10^-3 (for the critic), a weight decay of 0, max episodes of 1500, max timesteps of 300. The buffer size is how large the replay buffer is, the batch size is how large of a sample is pulled from the buffer, the gamma is the discount factor for how far in the future the reward is, tau is how much the weights of the target network are updated by (from the local network), learning rate is how quickly the model converges / does not converge, and weight decay is a regularization parameter.

Architecture

The architecture is state size to 30 hidden nodes with relu activation to 50 hidden nodes with relu activation to action size for the actor (with a tanh function applied at the end). The architecture is state size to 30 hidden nodes with relu activation to a concatenation of action space size to 50 hidden nodes with relu activation to action size for the critic (with a relu function applied at the end).

Rewards per Episode

Episode was solved in 235 episodes! Detailed graph of total rewards over each episode below.



Future Work

To be completely fair, the results of this project surprised me. The hidden nodes included in this project was in the tens sizes lower than my previous projects. I definitely think that it would be worth it to try very small amounts of hidden nodes, much more hidden nodes, or running the program until it converges and track the progress of the draft.

There was a suggestion that I did not end up implementing, which was to make the critic consider all states (both rackets) when giving its expected value. This could definitely improve the rate at which the environment would be solved if it works as intentioned.

Another idea (which might not be practical) would be to run a few environments at the same time with all of the agents sharing the same agent object and model. With a strong enough computer, I believe that this could come up with results that could overtake the quick learning process of the model I show here.