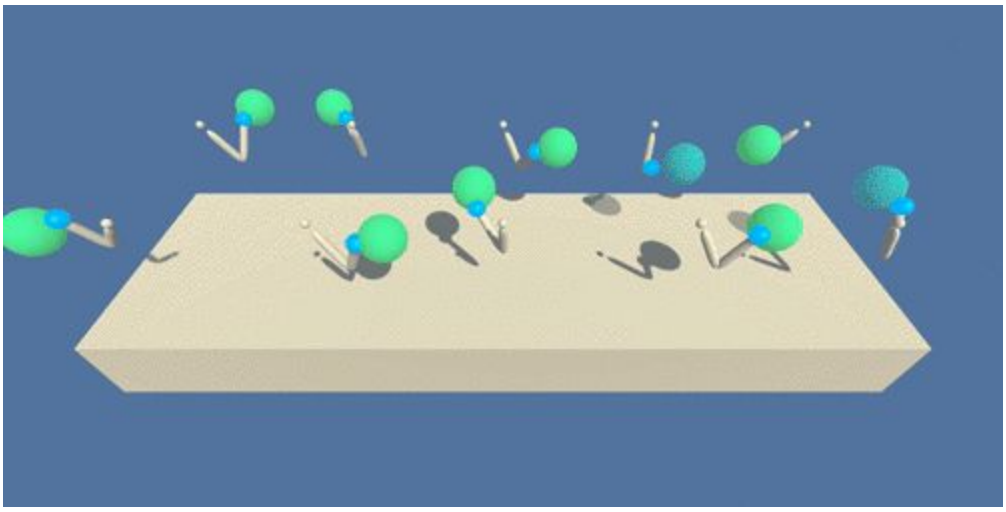
Project 2: Continuous Control

Jonathan Sullivan



For this project, I trained an agent in a Jupyter notebook to control a double-jointed arm in the Reacher environment. Given state information, the agent learned how to best select actions. The goal of my agent is to maintain its position at the target location for as many time steps as possible.

Environment

The task is episodic, and in order to solve the environment, my agent had to get an average score of +30 over 100 consecutive episodes.

Reward

A reward of +0.1 is provided for each step that the agent's hand is in the goal location.

State

The state space has 33 variables corresponding to position, rotation, velocity, and angular velocities of the arm.

Actions

Each action is a vector with four numbers, corresponding to torque applicable to two joints. Every entry in the action vector should be a number between -1 and 1.

The Learning Algorithm

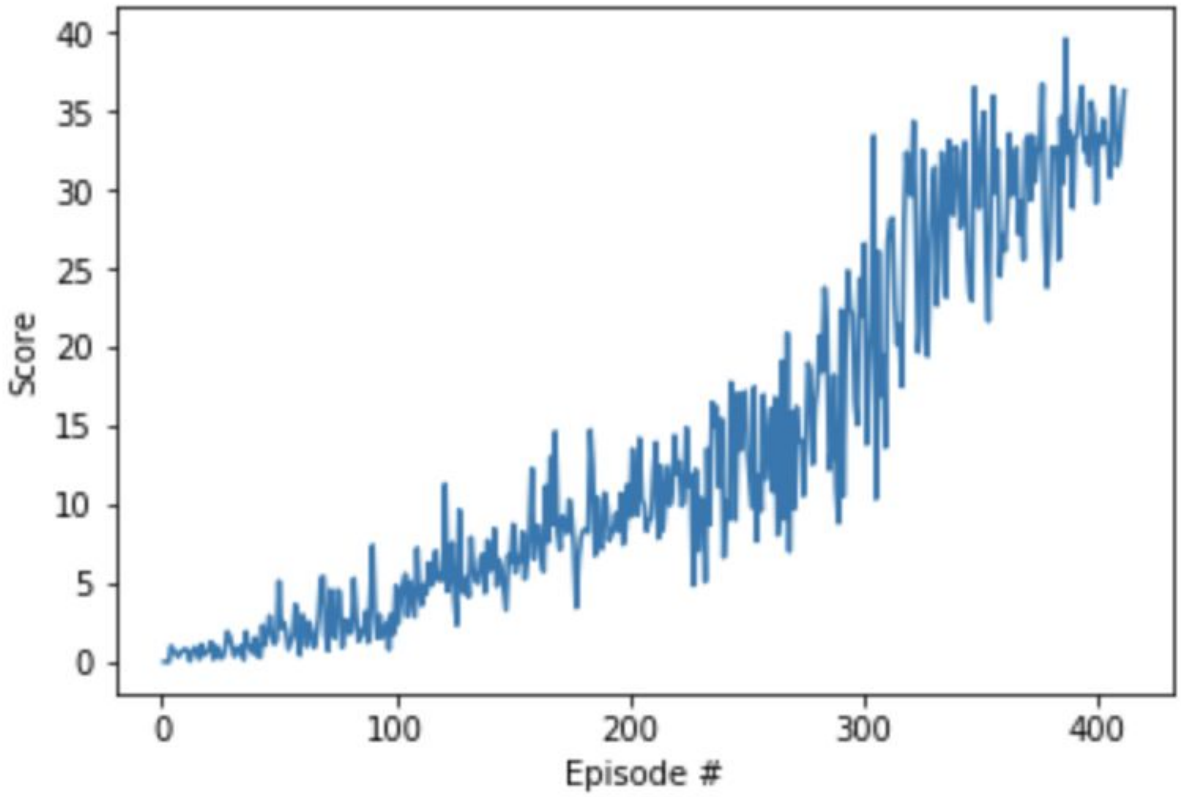
Background: Actor-Critique methods

Deep learning agents that use a deep neural network to approximate a value-function, such as state-Value, advantage-function or action-Value, the agent is said to be value-based(ie. DQN). However deep learning agents that use a deep neural network to approximate a policy, deterministic or stochastic, the agent is said to be policy-based(ie. REINFORCE). The Actor-Critic methods is an intersection of policy-based methods and value-based methods. Actor-Critic methods use value-based techniques to further reduce variance of policy-based methods. The actor (policy-based method) is used to find a good policy by examining what action it took to reach a goal and learning from those experiences. However this process in practice leads to high variance, since some good action might have been taken in an episode that lead to bad results or vice versa. While given an infinite amount of time the agent will converge this normally means slow training times. The critic in on the other hand is used to make guesses about about expected reward. Since the world is not known to the agent at the beginning of the problem the estimation will be bad but as learing continues it becomes better and better. This process however introduces bias since our estimate will be prone to over and underestimation. However these estimates are a lot more stable than the estimated policy of our actor, which has much less bias than our critic. So the actor learns to act and the critic learns to estimate situations and actions. Actor Critic agents learn by playing games and adjusting the probability of good and bad actions just with one actor alone. But the critic is able to allow our agent to tell good and bad action apart more quickly and speed up learning. Actor-Critic methods yield to better results with low variance and low bias, being more stable than value based agent and requiring few examples than policy based agents.

DDPG

The algorithm I used is called the Deep Deterministic Policy Gradient, DDPG. DDPG is an Actor-Critic method or approximate DQN since the critic in ddpg is used to approximate the maximizer over the Q-values of the next state and not as a learned baseline. DDPG is especially useful in this case because unlike DQN's it can be used for problems with continuous action spaces, such as "how much torque to apply in a rotational joint". This is because the actor in DDPG is used to approximate the deterministic optimal policy. Therefore we always choose the best believed action and not take action statistically. The actors learns the best action, while the Critic learns to evaluate the evaluate the optimal action-value function by using the actor's best perceived action, similar to how DQN's do.

Like DQN's, DDPG use replay buffers to break correlation among consecutive experience tuples and soft update to target network to prevent overestimation. Unlike my DQN implementation the target network is not fixed for a specified period of time and then updated. Instead, DDPG slowly blending the local network weights with the target network weights after each timestep. This update strategy lead to faster convergence than the tradition wait n-steps and update approach.



Future Improvements

Even though my algorithm solved the given environment. There are various improvements that I could have made. I could have used to make learning more stable and efficient is Prioritized Experience Replay. This ensures that older and rarer experience influence learning as much as the younger more frequent counterparts.

Another technique that I could have used to make learning more faster is using parallel agents. A group of agents share a singular target networks and execute separate identical tasks would lead to all of them sharing their experiences or at least the weight-values it has learned from the experiences causing faster convergence to the optimal policy and optimal action-value function.