Deep Reinforcement Learning Continuous Control Project

Project Overview

The goal of this project is to train an agent in a Unity Reacher environment. The agent is a double-jointed arm which can move to target locations. A reward of +0.1 is provided for each step that the agent's hand is in the goal location. The goal of the agent is to maintain its position at the target location for as many time steps as possible.

The observation space consists of 33 variables corresponding to position, rotation, velocity, and angular velocities of the arm. 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 environment is considered as solved if the average reward is larger then 30 for 100 episodes.

For this project, two separate versions of the Unity environment are provided:

-The first version contains a single agent.
-The second version contains 20 identical agents, each with its own copy of the environment.

This report is about training with the first environment with a single agent although I tried the same notebook for the second environment which I submit with the project.

Algorithms and Techniques

To solve the environment we use Actor Critic Methods which combines value-based and policy-based methods to bring together the best of two worlds. It uses value-based techniques to further reduce the variance of the policy-based methods. Actor-Critic agents are usually more stable than value-based agents and need fewer samples than policy-based agents. Actor-Critic agents use 2 neural networks, one for Actor and one for Critic. The Critic will learn to evaluate the state-value function using TD estimate. Using the Critic we will calculate the advantage function and train the Actor using this value. Some of the most popular Actor-Critic agents are A3C, A2C, GAE and DDPG. For this project I used a Deep Deterministic Policy Gradients or DDPG agent.

DDPG can be considered as an approximate DQN instead of a natural Actor-Critic. The Critic in DDPG is used to approximate the maximizer over the Q values of the next state. The Actor is used to approximate the optimal policy deterministically. It always gives the best believed action for any given state. The Critic learns to evaluate the optimal action value function by using the Actor’s best believed action. 2 other aspects of DDPG are using the replay buffer and using soft updates to the target networks. In soft update, the target network is updated from the local network by slowly blending the local network weights.

I used the code sample from Udacity classroom content, specifically from Bipedal and Pendulum model and DDPG agent python code as baseline. The Actor consists of 3 linear layers. The first 2 linear layers are followed by Relu layer and the 3rd linear layer is followed by a Tanh layer as output layer. The sizes for the layers are 33*400, 400*300 and 400*4 respectively. The Critic has a similar structure with sizes of 33*400, 404*300 and 300*1.

DDPG agent uses the above Actor and Critic classes and also uses ReplayBuffer class to save and reuse experiences and OUNoise class to be used to create the next action.

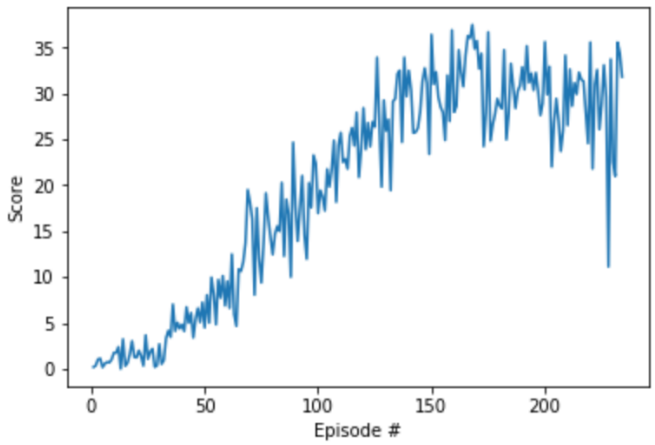
I tried training while changing some of the hyper parameters specifically LR_ACTOR and LR_CRITIC. Also I did some simple changed to Agent class to get number of agents as an argument and add experience to Replay buffer from arbitrary number of agents.

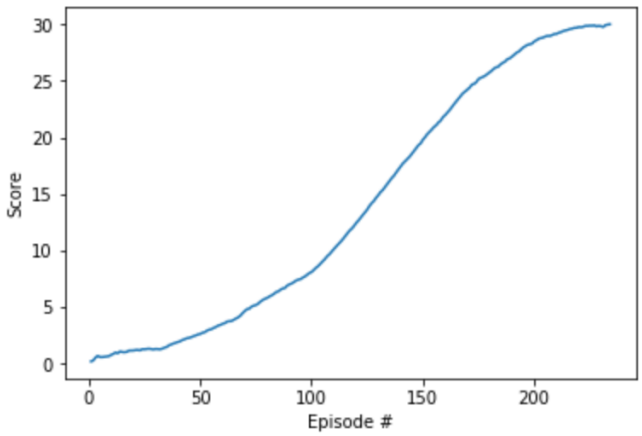
I trained an agent with the environment with 1 agent and saved the network weights as **checkpoint_actor_reacher.pth** and **checkpoint_critic_reacher.pth**. Also I trained agents with the environment with 20 agents and saved the network weights as **checkpoint_actor_reacher20.pth** and **checkpoint_critic_reacher20.pth**.

Training results

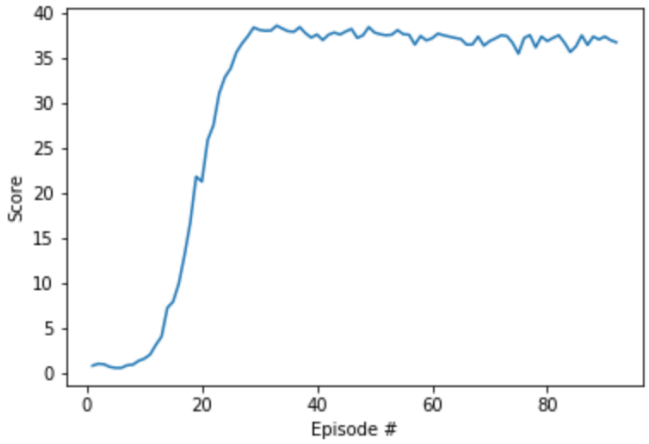
Using the same Actor and Critic networks and same Agent and hyper parameters I trained the 2 environments and here's the results and graphs:

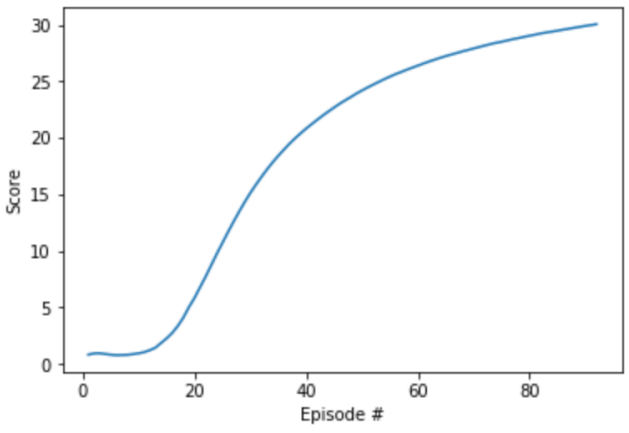
For environment with 1 agent, the agent was trained in 234 episodes with an average of 30 for 100 episodes. The graphs for episodes scores and average scores are as follows:





For the environment with 20 agents the agent was trained in 92 steps and the graphs for episodes scores and average scores are as follows:





The training needed fewer episodes for environment 2 as expected because more shared experiences gathered in fewer episodes. Running the code many times, the environment with 1 agent sometimes took too many steps and did not converge but the environment with 20 agents always converged with reasonably small number of episodes.

Ideas for Future Work

It would be ideal if there is a training algorithm which can train an agent for a class of problems without changing the parameters or network size. I tried the same Actor and Critic network and Agent to train an agent for Crawler environment. I ran it for 6000 episodes and although the agent learned to walk for more than 100 steps it was still not walking very well.

I'd like to use other Actor Critic agents like A3C, A2C, GAE and combine some of the good ideas from each to see if I can find a "model free" network which can solve a good number of continuous control environments without changing parameters.