DEEP REINFORCEMENT LEARNING PROJECT 2 – CONTINUOUS CONTROL

The goal of this project is to train a double-jointed arm (agent) to maintain its position at a moving target location for as many steps as possible. A reward of +0.1 is provided for each step that the agent's is at the goal location. The environment state space has 33 dimensions corresponding to the position, rotation, velocity, and angular velocities of the arm. The action space consists of four continuous numbers [-1, 1] corresponding to the torque applied to two joints. The task is episodic and is solved when an average score of +30 (over 100 consecutive episodes and over all agents) is achieved.

LEARNING ALGORITHM

I used the Deep Deterministic Policy Gradient (DDPG) algorithm to solve the second version of the environment (20 agents) because of its excellent performance on continuous control problems [1]. The DDPG algorithm learns a policy by employing actor and critic neural networks. The actor determines the next action to take given the environment's state as input while the critic predicts the value ($Q(s, a)$) of the action taken in that state. The actor is trained using a sampled policy gradient (which is derived from the Q-values produced by the critic) while the critic is trained to minimize the one-step TD-Error (which is derived from the state, action, reward, next state produced by the actor interacting with the environment).

ACTOR-CRITIC NEURAL NETWORK ARCHITECTURE

I started from Udacity's DDPG implementation [2] for the OpenAI Gym's Pendulum environment and evaluated multiple hyperparameter settings (e.g. learning rates and hidden layer sizes) until I solved the environment.

Actor Neural Network

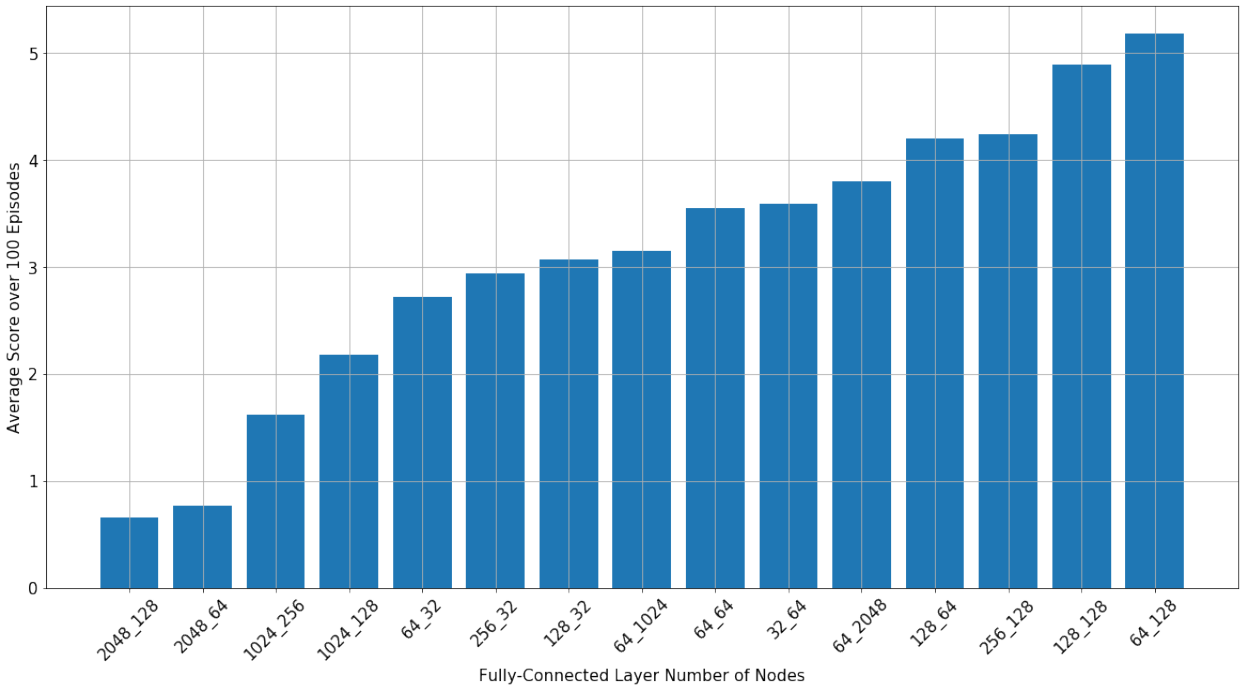
The actor used three fully-connected layers. The first layer had 64 hidden nodes, a ReLU activation function, and batch normalization. The second layer had 128 hidden nodes and ReLU activation function. The third layer had 4 hidden nodes (corresponding to the action-space size) and a Tanh activation function (to produce actions between -1 and 1). The actor was trained with a learning rate of 2.6e-3 and batch size of 128.

Critic Neural Network

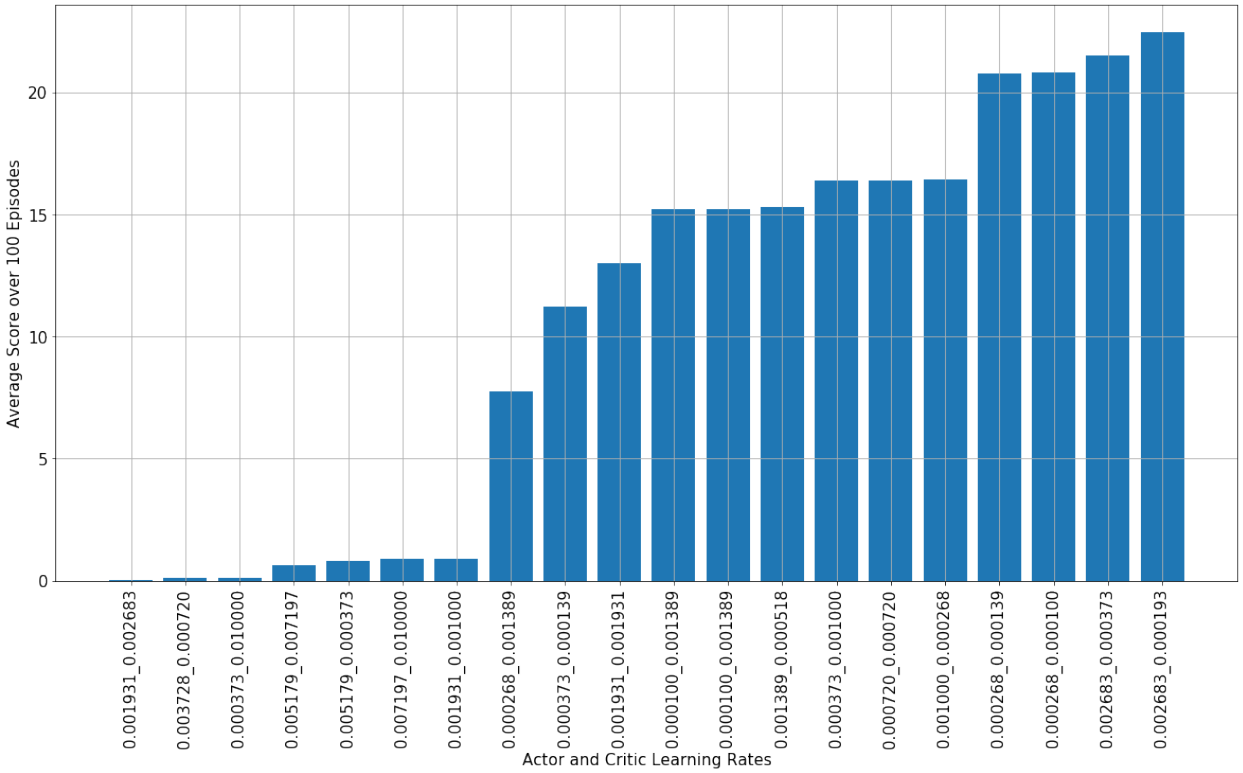
The critic used three fully-connected layers. The first layer had 64 hidden nodes, a ReLU activation function, and batch normalization. The second layer had 128 hidden node and ReLU activation function. The third layer had 1 hidden node and a no activation function (to produce a single q-value). The critic was trained with a learning rate of 1.9e-4 and batch size of 128.

DETERMINING NEURAL NETWORK HYPEPARAMETERS

The plot below illustrates the mean score over 100 episodes for agents with different combinations of first and second fully-connected (FC) layer sizes in the actor and critic networks. More specifically, the x-label "2048_128" means 2048 hidden nodes in both the actor and critic's first FC layer and 128 hidden nodes in both the actor and critic's second FC layer. Based on this result, I selected 64 hidden nodes in the first FC layer of the actor and critic and 128 nodes in the second FC layer of the actor and critic.



With the first and second FC layer sizes of both networks fixed at 64 and 128 respectively, I then evaluated the mean score over 100 episodes for agents with different learning rate combinations for the actor and critic networks. More specifically, the x-label "0.001931_0.002683" means a learning rate of 1.9e-3 for the actor and a 2.6e-3 for the critic. Based on this result, I selected a learning rate of 2.6e-3 for the actor and 1.9e-4 for the critic.



TRAINING WITH MULTIPLE AGENTS

Two techniques were employed to help the 20 agents jointly learn.

Sharing Experiences

A shared replay buffer (of size 5e6) accumulated the experiences (state, actions, reward, next state) of all the agents. From this replay buffer all agents sampled mini-batches during training.

Sharing Parameters

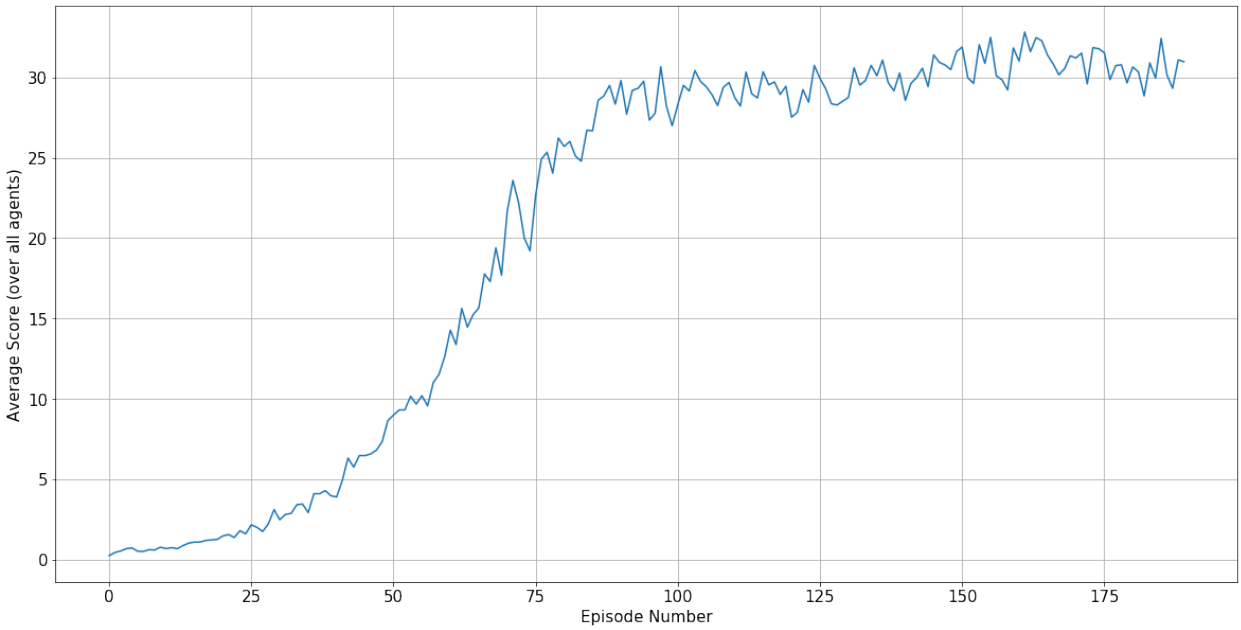
The network parameters of the best agent where shared with agents performing poorly to prevent those agents from getting stuck. More specifically, at the end of each episode, agents were ranked based on their mean score over the previous 100 episodes. Then the parameters of the best agent replaced those of agents with a mean score more than 1 point away from that of the current best agent. Otherwise, no parameter exchange took place between agents.

Decrease Frequency of Learning

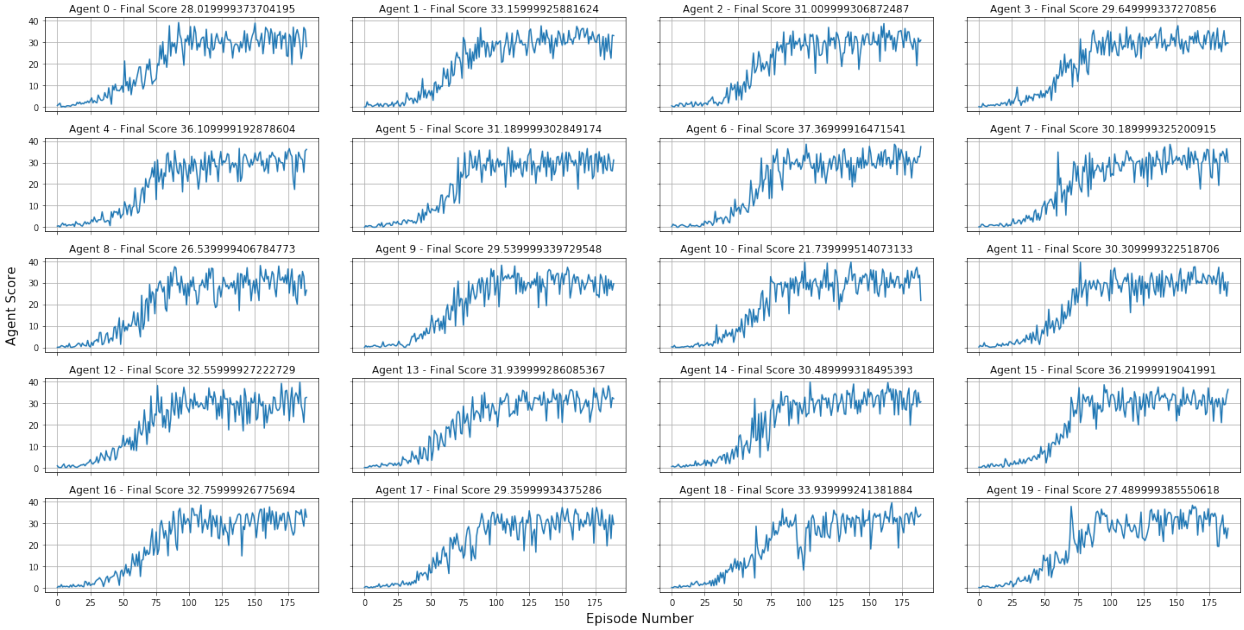
As noted in the project description, it is also beneficial to decrease the number of updates made to the network. So, rather than update the network at every timestep, agents should interact with the environment for a while (e.g. 20 timesteps) and then take a few learning steps (e.g. 10 steps).

RESULTS

The agents achieved an average score (over 20 agents and the most recent 100 episodes) greater than 30 (30.0005) after 190 episodes. The figure below illustrates the evolution of the score (averaged over 20 agents) and illustrates how learning accelerates after 25 episodes and slows down after 100 episodes.



The figure below illustrates the evolution of the score for each agent in the 20-agent pool as well as each agent's score at episode 190 (Final Score). Agent 6 achieved a score of 37.4 on the final episode.



FUTURE WORK

Future work on this project would proceed in two directions. The first would involve continuing with the DDPG algorithm but exploring different neural network architectures for the actor/critic (e.g. adding more layers and using other activation functions) and further exploring how agents can share their learning beyond simply sharing experiences using a shared replay buffer. The second direction would involve trying out different RL algorithms such as Trust Region Policy Optimization (TRPO).

REFERENCES

[1] Lillicarp et al. *Continuous Control with Deep Reinforcement Learning*. arXiv:1509.02971v5 (link)

[2] Udacity Deep Reinforcement Learning Github (link)