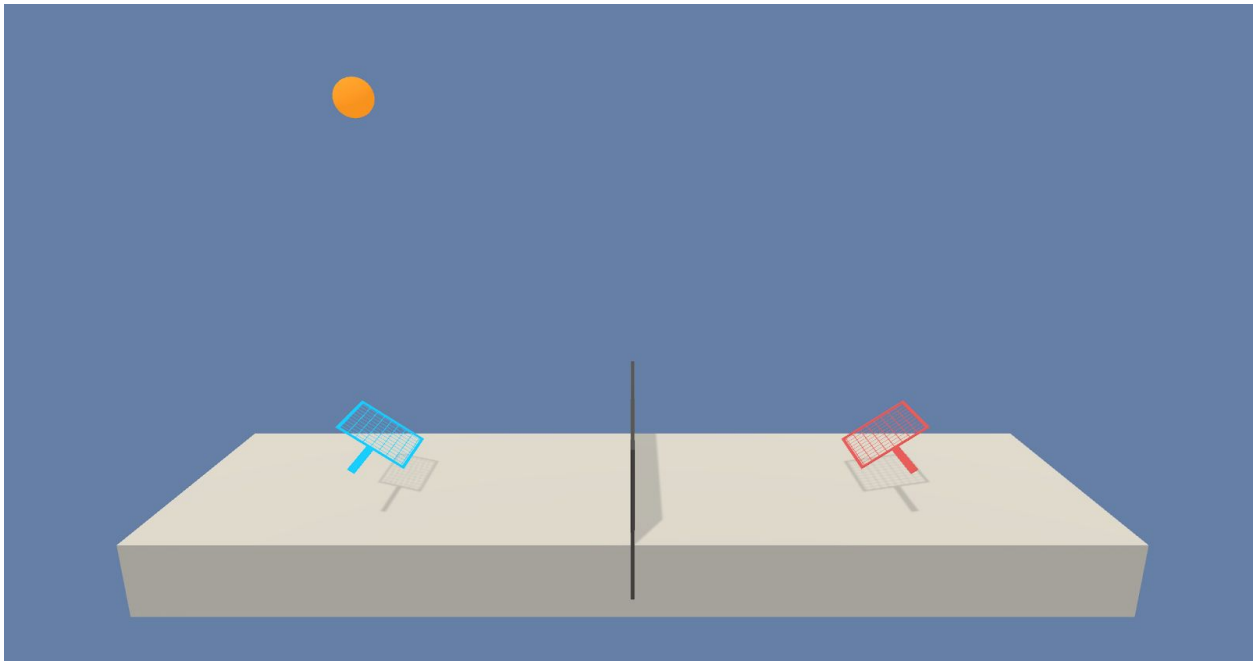
Project 3: Collaboration and Competition

Jonathan Sullivan



In this environment, two agents control rackets to bounce a ball over a net.

For this project, I trained two agent in a Jupyter notebook to control rackets to bounce a ball over a net in the Tennis environment. Given state information, each agent learned how to best select actions. The goal of each agent is to keep the ball in play.

Environment

The task is episodic, and in order to solve the environment, my agents had to get an average score of +0.5 over 100 consecutive episodes after taking the maximum over both agents.

Reward

A reward of +0.1 is provided if an agent hits the ball over the net and a reward of -0.01 is provided if an agent lets a ball hit the ground or hits the ball out of bounds. After each episode, we add up the rewards that each agent received (without discounting), to get a score for each agent. This yields 2 (potentially different) scores. We then take the maximum of these 2 scores. This yields a single score for each episode.

State

The state space has 8 variables corresponding to the position and velocity of the ball and racket. Each agent receives its own, local observation.

Actions

Each action is a vector with 2 numbers, corresponding to movement toward (or away from) the net, and jumping. 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 learning 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.

Background: 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 force to apply to hitting a ball". 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

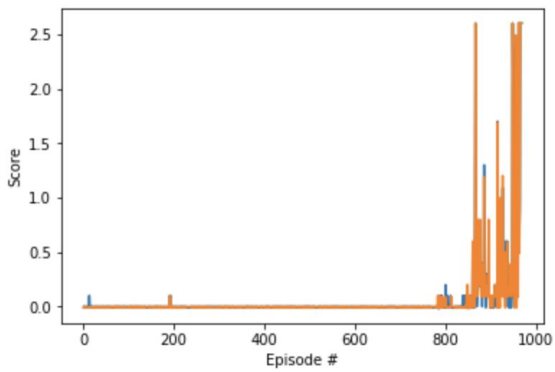
the evaluate the optimal action-value function by using the actor's best perceived action, similar to how DQN's do.

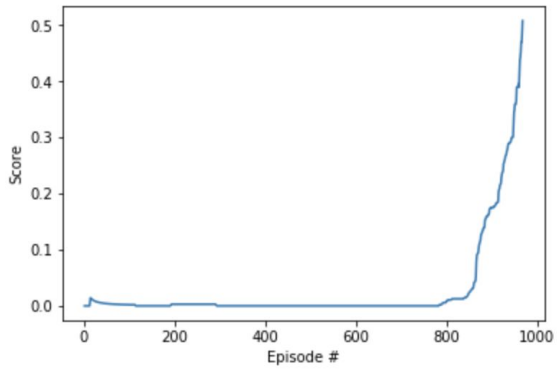
Like DQN's, DDPG use replay buffers to break correlation among consecutive experience tuples and soft update to target network to prevent overestimation. Unlike my DQN implementation the target network is not fixed for a specified period of time and then updated. Instead, DDPG slowly blending the local 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.

MADDPG

The algorithm I used is called the Multi Agent Deep Deterministic Policy Gradient, MADDPG. MADDPG is an extension of the DDPG. In MADDPG there is more than one agent. These agent may interact in ways that are cooperative, competitive or a mixture of both. These agents can also be trained independently or together sharing the same network.

In my implementation I trained two DDPG agents to act cooperatively. They are train on identical independent neural networks and have a shared replay buffer. The network I used for the actor contained an input layer of size 33 and an output layer of size 2. This network has 2 hidden layers with rectified linear unit activations and the output layer has a hyperbolic tangent activation function. The network I used for the critic contained an input layer of size 34 and an output layer of size 2. This network has 2 hidden layers with rectified linear unit activations and the output layer. In the input layer of the critic we inject the action provided by the actor. The hyperparameter I chose were as follows. The replay buffer size was 100000. The minibatch size for training the networks were 250. The discount factor for calculating return was 0.99. The TAU-value for soft update of target parameters: .005. The learning rate for the actor was .0001 and the learning rate for the critic was .001. The L2 weight decay was 0.





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 pair-agents share a singular target networks and executing 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.