Deep Reinforcement Learning – Project 3: Collaboration and Competition

NIKHIL TIWARI

We work with the tennis environment for this project. In this environment, two agents control rackets to bounce a ball over a net. If an agent hits the ball over the net, it receives a reward of +0.1. If an agent lets a ball hit the ground or hits the ball out of bounds, it receives a reward of -0.01. Thus, the goal of each agent is to keep the ball in play. The observation space is 24 dimensional consisting of 8 variables corresponding to the position and velocity of the ball and racket. Each agent receives its own, local observation. Two continuous actions are available, corresponding to the movement toward (or away from) the net, and jumping.A two-agent tennis environment with 24-dimensional state-space.The task is episodic, and in order to solve the environment, your agents must get an average score of +0.5 (over 100 consecutive episodes, after taking the maximum over both agents). Specifically, 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. The environment is considered solved when the average (over 100 episodes) of those scores is at least +0.5.

Algorithms

Deep Deterministic Policy Gradients

Deep Deterministic Policy Gradient (DDPG)[1] lies under the class of Actor-Critic Methods but is a bit different than the vanilla Actor-Critic algorithm. The actor produces a deterministic policy instead of the usual stochastic policy and the critic evaluates the deterministic policy. The critic is updated using the TD-error and the actor is trained using the deterministic policy gradient algorithm. In fact, it could be seen as an approximate Deep-Q Network(DQN) method. The reason for this is that the critic in DDPG is used to approximate the maximizer over the Q-values of the next state, and not as a learned baseline, as have seen so far. One of the limitations of the DQN agent is that it is not straightforward to use in continuous action spaces. For example, how do you get the value of continuous action with DQN architecture? This is the problem DDPG solves.

Replay Buffer

In reinforcement learning, the agent interacts with the environment and learns from the sequence of state(S), actions(A) and rewards(R) and next state(S') which is stored in the form of an experienced tuple. These sequences of experience

tuples can be highly correlated, and the DDQN algorithm like the Q-learning algorithm that learns from each of these experienced tuples in sequential order can be swayed by the effects of this correlation. This can lead to oscillation or divergence in the action values, which can be prevented by having a replay buffer, from which experience replay tuples are used to randomly sample from the buffer. The replay buffer contains a collection of experience tuples (SS, AA, RR, S'S ′). The tuples are gradually added to the buffer as we are interacting with the environment. The act of sampling a small batch of tuples from the replay buffer in order to learn is known as experience replay. In addition to breaking harmful correlations, experience replay allows us to learn more from individual tuples multiple times, recall rare occurrences, and in general make better use of our experience.

Better Implementation and Approach

The main components of DDPG have been explained in the previous report. In this report,we will focus on the modification made on the standard DDPG to solve the Tennis environment.

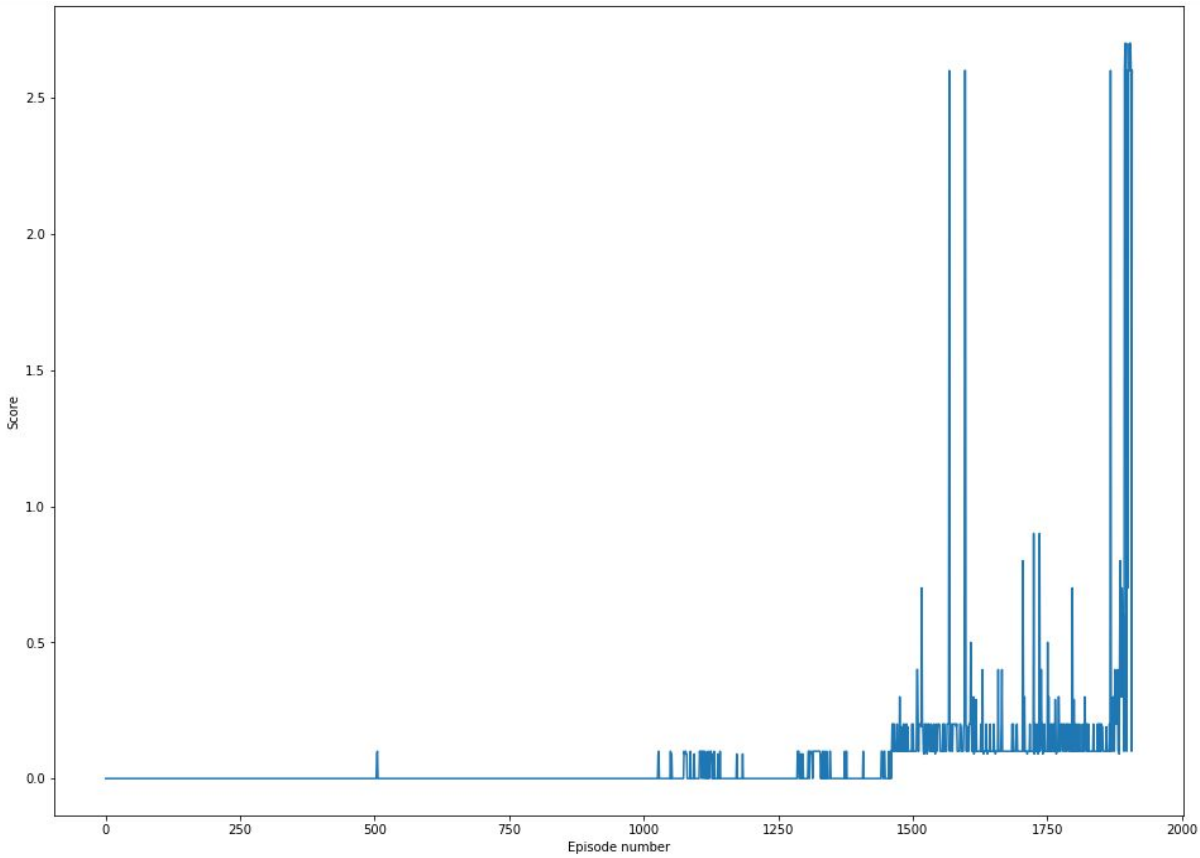
The main difference is that all agents share the same replay buffer memory. The main reason is that the agent may get caught in a tight loop of specific states that loop back through one another and detract from the agent exploring the full

environment “equally”. By sharing a common replay buffer, we ensure that we explore all the environment. Note that the agents still have their own actor and critic networks to train.

Because we have two models for each agent; the actor and critic that must be trained, it means that we have two sets of weights that must be optimized separately. Adam was used for the neural networks with a learning rate of 10^{-4} and 10^{-3} respectively for the actor and critic for each agent. For Q, I used a discount factor of $\gamma = 0.99$. For the soft target updates, the hyperparameter τ was set to 0.001. The neural networks have 2 hidden layers with 250 and 100 units respectively. For the critic Q, the actions did not include the 1st hidden layer of Q. The final layer weights and biases of both the actor and critic were initialized from a uniform distribution $[-3 \times 10^{-3}, 3 \times 10^{-3}]$ and $[3 \times 10^{-4}; 3 \times 10^{-4}]$ to ensure that the initial outputs for the policy and value estimates were near zero. As for the layers, they were initialized from uniform distribution $[-1/\sqrt{f}, 1/\sqrt{f}]$ where f is the fan-in of the layer.

Results

The agents are trained until they solve the environment, that is to say when one the two agents obtains an average reward of at least +0.5 over the last 100 episodes. **The environment is solved in 1809 episodes !**



Plot of scores per episode

Future Ideas:

As such, there is no really such adversarial or collaborative relationship needed, thus the multi-agent reinforcement learning does not seem ideal in this setting. A great challenge would be to tackle an environment in which the Multi-Agent Reinforcement Learning framework could apply, like the Soccer environment. With it, we can apply Multi-Agent DDPG[2] in which each of the actors takes a concatenation of all the states and actions of the other agents.

References:

[1] DDPG Paper: https://arxiv.org/abs/1509.02971
[2] Multi-Agent DDPG: https://papers.nips.cc/paper/7217-multi-agent-actor-critic-for-mixed-cooperative-competitive-environments.pdf