Udacity Deep Reinforcement Learning

Project: Multi Agent Reinforcement Learning

Author – Anish Amul Vaidya (avaidya172@gmail.com)

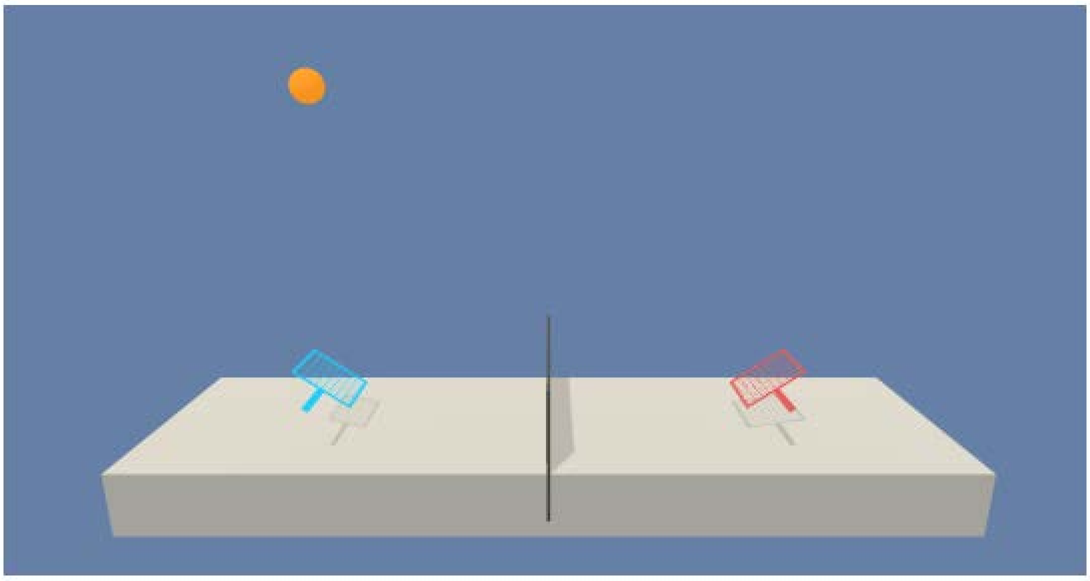
Abstract - In this environment, two agents control rackets to bounce a ball over a net. If an agent hits the ball over the net, it receives a reward of +0.1. If an agent lets a ball hit the ground or hits the ball out of bounds, it receives a reward of -0.01. Thus, the goal of each agent is to keep the ball in play.

The observation space consists 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 movement toward (or away from) the net, and jumping.

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.

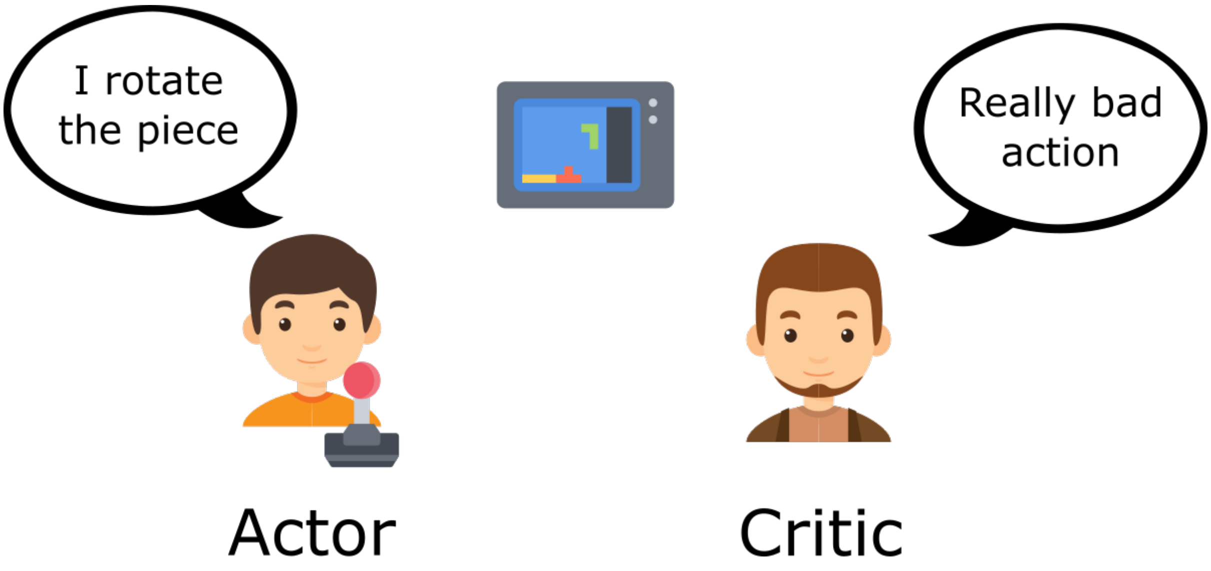


The environment is considered solved, when the average (over 100 episodes) of those scores is at least +0.5.The following report is written in four parts:

- Implementation
- Results
- Possible future improvements

Implementation

The basic algorithm lying under the hood is an actor-critic method. Policy-based methods like REINFORCE, which use a Monte-Carlo estimate, have the problem of high variance. TD estimates used in value-based methods have low bias and low variance. Actor-critic methods marry these two ideas where the actor is a neural network which updates the policy and the critic is another neural network which evaluates the policy being learned which is, in turn, used to train the actor.



In vanilla policy gradients, the rewards accumulated over the episode is used to compute the average reward and then, calculate the gradient to perform gradient ascent. Now, instead of the reward given by the environment, the actor uses the value provided by the critic to make the new policy update.

Policy Update: $$\Delta\theta = \alpha * \nabla_\theta * (log\, \pi(S_t, A_t, \theta)) * \cancel{R(t)}$$

New update: $$\Delta\theta = \alpha * \nabla_\theta * (log\, \pi(S_t, A_t, \theta)) * \boxed{Q(S_t, A_t)}$$

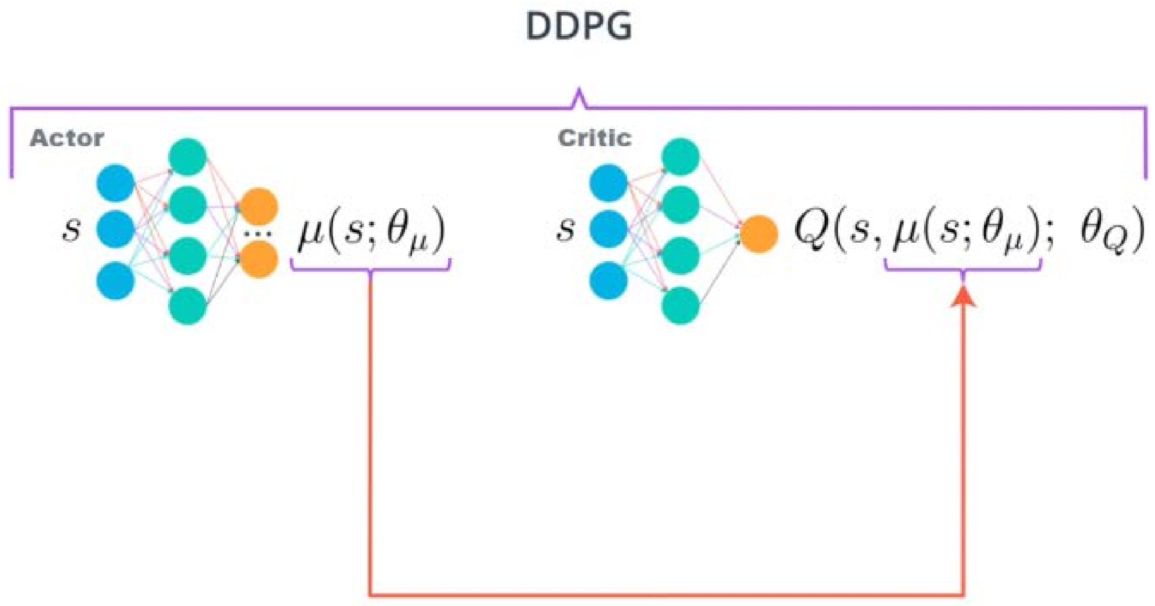
Deep Deterministic Policy Gradient (DDPG) lies under the class of Actor Critic Methods but is a bit different than the vanilla Actor-Critic algorithm. The actor produces a deterministic policy instead of the usual stochastic policy and the critic evaluates the deterministic policy. The critic is updated using the TD-error and the actor is trained using the deterministic policy gradient algorithm.

$$\begin{aligned}\nabla_{\theta^\mu} J &\approx \mathbb{E}_{s_t \sim \rho^\beta}\left[\nabla_{\theta^\mu} Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t|\theta^\mu)}\right] \\ &= \mathbb{E}_{s_t \sim \rho^\beta}\left[\nabla_a Q(s, a|\theta^Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta_\mu} \mu(s|\theta^\mu)|_{s=s_t}\right]\end{aligned}$$

- **Fixed targets**: Originally introduced for DQN, the idea of having a fixed target has been very important for stabilizing training. Since we are using two neural networks for the actor and the critic, we have two targets, one for actor and critic each.

- **Soft Updates**: In DQN, the target networks are updated by copying all the weights from the local networks after a certain number of epochs. However, in DDPG, the target networks are updated using soft updates where during each update step, 0.01% of the local network weights are mixed with the target networks weights, i.e. 99.99% of the target network weights are retained and 0.01% of the local networks weights are added.
- **Experience Replay**: This is the other important technique used for stabilizing training. If we keep learning from experiences as they come, then we are basically observed a sequence of observations each of which are linked to each other. This destroys the assumption of the samples being independent. In ER, we maintain a Replay Buffer of fixed size (say N). We run a few episodes and store each of the experiences in the buffer. After a fixed number of iterations, we sample a few experiences from this replay buffer and use that to calculate the loss and eventually update the parameters. Sampling randomly this way breaks the sequential nature of experiences and stabilizes learning. It also helps us use an experience more than once.

DDPG(Lilicrapetal.,2015), short for Deep Deterministic Policy Gradient, is a model free off-policy actor-critic algorithm, combining DPG with DQN. Recall that DQN (Deep QNetwork) stabilizes the learning of the Q-function by using experience replay and a frozen target network. The original DQN works in discrete space, and DDPG extends it to continuous space with the actor-critic framework while learning a deterministic policy. In DDPG, we use 2 deep neural networks : one is the actor and the other is the critic:

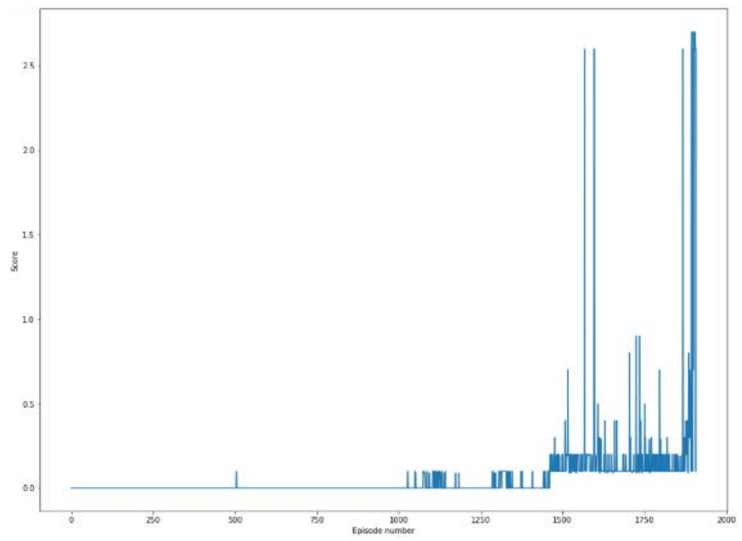


There is a modification to the DDPG agent used in the 'continuous control - robot arm' project. The main difference is that here, all the agents share the same replay buffer memory. The main reason that the agent may get caught in a tight loop of specific states that loop back through one another and detract from the agent exploring the full environment "equally". By sharing a common replay buffer, we ensure that we explore the whole environment. Both the agents still have their own actor 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 set 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 γ = 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 were not included the 1st hidden layer of Q. The final layer weights and biases of both the actor and critic were initialized from a uniform distribution $[−3×10^{-3},3×10^{−3}]$ and $[3×10^{−4};3×10^{−4}]$ 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 / √f , 1 / √f] where f is the fan-in of the layer. We use the prelu1 activation function for each layer.

Results

The following is the graph of the scores per episode as the models train:



Possible future improvements

Because of the symmetry of the environment, each agent mirrors the other one behaviour. As such, there is no really such adversarial or collaborative relationship needed, thus the multiagent 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 (Lowe et al.) in which each of the actor takes a concatenation of all the states and actions of the other agents.