Multi-Agent Continuous Reinforcement Learning Problem Report

In this exercise, I used a Multi Agent Deterministic Deep Policy Gradient (MADDPG) Agent to control a unity environment brain, that in turns controls multiple agents in a Tennis game. In this report, I documented the characteristics of the environment, the method used to solve the environment control problem, the result of the training process, and what can be done to improve it.

The Environment

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 Agents

During this exercise I used the combination of two algorithm to represent and train the behavior of the agents. The agents individually were modeled after the DDPG Deterministic Deep Policy Gradient agent and their training behavior was modeled as MADDPG Multi Agent Deterministic Deep Policy Gradient agents. Follow the description of each algorithm

DDPG Agent

Each agent is a version of a DDPG (https://arxiv.org/pdf/1509.02971) that uses two deep neural network the actor represents the agent policy, the critic calculates the q_value of the actions taken by the actor.

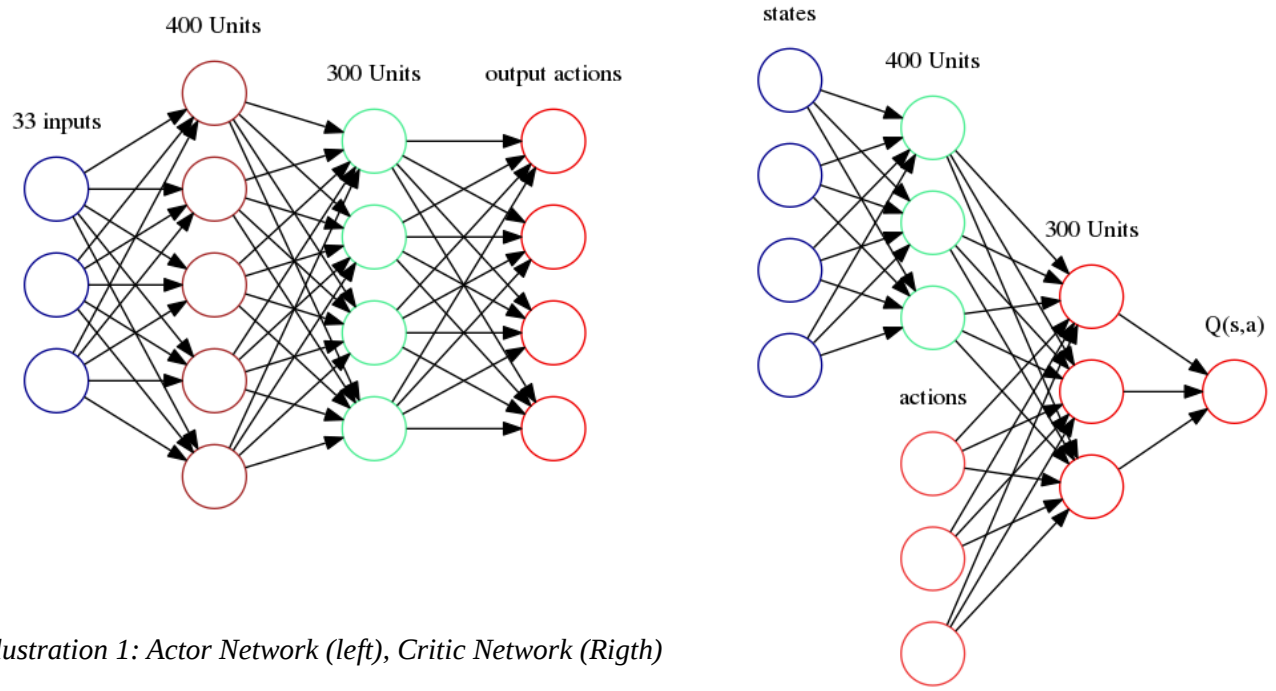


Illustration 1: Actor Network (left), Critic Network (Rigth)

The main idea behind the DDPG agents is the use on the Actor to determine directly the actions and their magnitude, that the agent has to apply in any given state. The role of the critic is used the action taken by the actor on a particular state and calculate the value of that action on that state.

MADDPG Agents

MADDPG or Multi-Agent Deterministic Deep Policy Gradient, is an algorithm used to train the DDPG agents, it is based on the “Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments” white paper (https://papers.nips.cc/paper/7217-multi-agent-actor-critic-for-mixed-cooperative-competitive-environments.pdf). As per the DDPG model, each agent posses two neural networks that represent them, one is the policy network that tell the agent what is the best action on a

giving state and the other is the critic network which is utilized only during training and determine the value of each action so the agent can learn to act based on the best possible action. The main idea behind MADDPG is that during training all agents should be aware of the other agents states if they are going to collaborate with each other or compete against each other. An example of such claim is that in a soccer game one player needs to pass the ball to other players to score, if player A does not know where his/her team player B is in the field he/she will not be able to accurately pass the ball. The same applies for competitive behavior, two boxers A and B needs to be aware of the rival spatial location in order to correctly land punches and defend them; this is the heart of MADDPG, each agent critic model is optimized with the state and actions of all the agent so the value of the action it calculates for its own agent actor is the best considering all the environment agents.

The Optimization

The Critic is optimized by taken the actions produced by all the environment actors their their next states and using the critic target network it calculates the next state q_next_state_value. From the experience buffer we extract the state, action, next_state and next_action, and using the Bellman equation (https://en.wikipedia.org/wiki/Bellman_equation) and the calculated q_next_state_value we calculate an estimate of the q_value for that particular agent on its current state; note that during this process the critic contains information of all the other agents. We use the difference of the q_value calculated using the Bellman equation (target) and the q_value produced by the critic local model (that also contains information about all the agents) as the loss or error signal for the critic network, in turn we used that loss to optimized the critic.

The Actor optimization process is based on the David Silver “Deterministic Policy Gradient” paper (http://proceedings.mlr.press/v32/silver14.pdf). The paper explains that the error signal of the Actor or agent policy function is in proportion of the sum of the negative of the gradient of the Critic q_values . In other words of each visited state “s”, the policy (actor_local neural network) parameters (weights) θ^{k+1} are updated in proportion to the gradient $\nabla_\theta Q^{\mu^k}(s, \mu_\theta(s))$. Each state suggests a different direction of policy improvement ; these may be averaged together by taking an expectation with respect to the state distribution.

In my code the above is implemented by calculating the average of the q_values using the states and the action returned by the policy, negating these average and back propagating the values to adjust the actor local model parameters (weights).; these method seems to work pretty good although I still trying to map the internal torch gradients calculating to the Davide Silver equations. I will modify this report to add more details once I have the map completed.

The parameters.

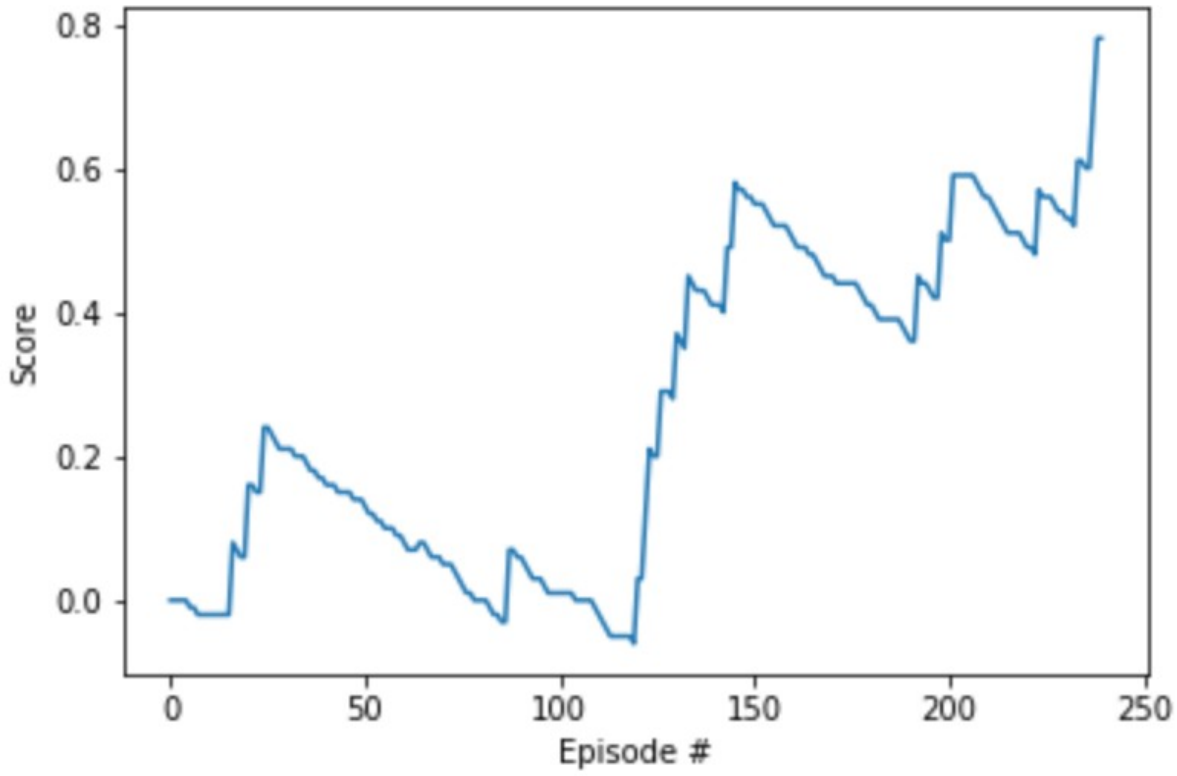
The replay buffer size was 1000000; the learning batch consisted of 128 samples. I used a discount factor of 0.95, to perform a soft update of the target weights (see the fix q-targets weight documentation in the dqn URL) I used a TAU factor of 0,02, my learning rate was of 0.0001 for the

actor and 0.001 for the critic, and I performed weighs update every time step, once I had at least 128 samples in the replay buffer.

The network was created using Pytorch.

The result of the training

After ~240 episodes the model solved the problem, with an averaged score over 100 steps greater than 0.5 and kept improving (see chart below)



Improvements

Even if the agent was training to an acceptable level, the following improvement could be made to optimize the agent behavior and make it converge to the option q* value in less training episodes.

Double Deep Q-Network

The problem with the dqn agent is that it tends to overestimate the q values, this is because it uses the q value if the actions that maximize the q estimate function to determine the real q value, this produces a tendency toward given values to actions higher than the real ones. The solution to this problem is the use of two sets of Q-target networks weighs, one is used to select the action and the other set of weighs is used to select the value of that action.

Prioritize Experience Replay

The main idea behind this improvement is that during the selection of the experiences to be used in the learning process, we select these using a random process, with this we run the risk of selecting the same set of samples or even worst not selecting valuable experiences. The improvement consists of using a parameter to weigh the importance of the experience base on its value. The value we can use to prioritize the selection of important experiences can be the error term obtained when we are performing the weigh optimization, using this value we can calculate the probability of the sample selection, the higher the experience error term the highest its probability is of been selected.