AI Project Report: Deep Deterministic Policy Gradient (DDPG) Algorithm for Tennis Unity ML-Agent Environment

Srikar Mukkamala *12141590*
Ashrith Vardhan *12140480*
Nikhil Reddy *12140630*
Ramesh Naik *12141090*

I. Introduction

This project implements the Deep Deterministic Policy Gradient (DDPG) algorithm to train two agents playing a game of Tennis against each other. The agents control rackets to bounce a ball over a net, aiming to keep the ball in play. The implementation uses a modified version of the Tennis Unity ML-Agent Environment and a Multi-Layer Feedforward Neural Network Model with the PyTorch library.

II. Problem Statement

The goal is for both agents to achieve an average score of +0.5 over 100 consecutive episodes, with the maximum score taken over both agents for each episode. The reward system includes +0.1 for hitting the ball over the net and a penalty of -0.01 for letting the ball hit the ground or hitting it out of bounds.

A. State and Action Spaces

The observation space consists of 8 variables representing the position and velocity of the ball and racket. Each agent receives its own local observation. The action space is continuous, with 2 continuous actions available for each agent, corresponding to movement toward (or away from) the net and jumping.

B. Solution Criteria

The environment is considered solved when both agents achieve an average score of +0.5 over 100 episodes.

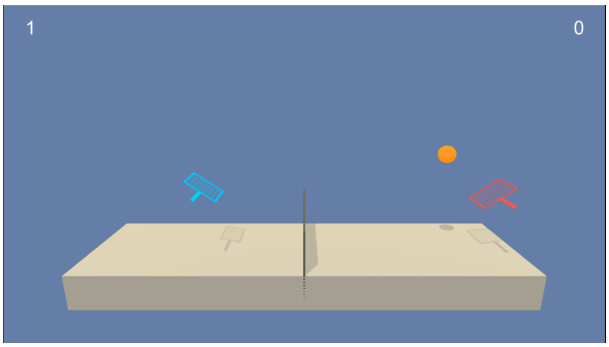


Fig. 1. Your Image Caption

III. Learning Algorithm

For this project, the Multi-Agent Deep Deterministic Policy Gradient (DDPG) algorithm was used to train the agent.

A. Deep Deterministic Policy Gradient (DDPG)

- DDPG concurrently learns a Q-function and a policy.
- It uses off-policy data and the Bellman equation to learn the Q-function and uses the Q-function to learn the policy.

- DDPG is an off-policy algorithm and can only be used for environments with continuous action spaces.
- DDPG can be thought of as being deep Q-learning for continuous action spaces.

This approach extends DDPG to support multiple agents, where both agents can access the observations and actions of each other and cooperate to keep the ball in play for as long as possible.

IV. Model Architecture and Hyperparameters

A. Actor

- Fully connected layer 1: Input 48 (state space), Output 512, RELU activation, Batch Normalization
- Fully connected layer 2: Input 512, Output 256, RELU activation
- Fully connected layer 3: Input 256, Output 2 (action space), TANH activation

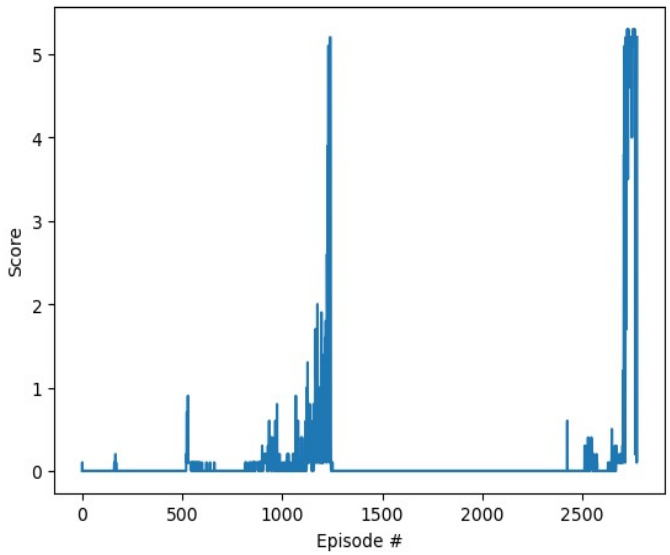
B. Critic

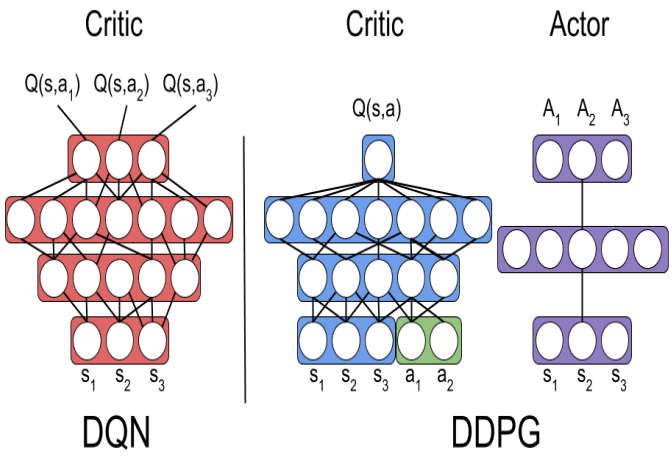
- Fully connected layer 1: Input 48 (state space), Output 512, RELU activation, Batch Normalization
- Fully connected layer 2: Input 512, Output 256, RELU activation
- Fully connected layer 3: Input 256, Output 1

V. Plot of Rewards

Below is a training run of the above model architecture and hyperparameters:

- Number of agents: 2
- Size of each action: 2
- Environment solved in 2300 episodes! Average Score: 3.15





Pseudocode

Algorithm 1 Deep Deterministic Policy Gradient

Input: initial policy parameters θ, Q-function parameters ϕ, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
repeat
  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for however many updates **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets

$$y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

      Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} (Q_\phi(s, a) - y(r, s', d))^2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$$

      Update target networks with

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1 - \rho)\phi$$
$$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1 - \rho)\theta$$

    end for
  end if
until convergence