Solving Unity Machine Learning Agents Tennis Environment Using Deep Deterministic Policy Gradient with Self-Play

Udacity - Project 3 Report: Collaboration & Competition

Ryan Herchig

1 Introduction

Multi agent reinforcement learning focuses on studying the behavior of multiple learning agents that coexist in a shared environment [1]. It consists of large number of artificial intelligence-based agents interacting with each other in the same environment, often collaborating towards a common end goal [2]. This project uses an environment in which 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 [3].

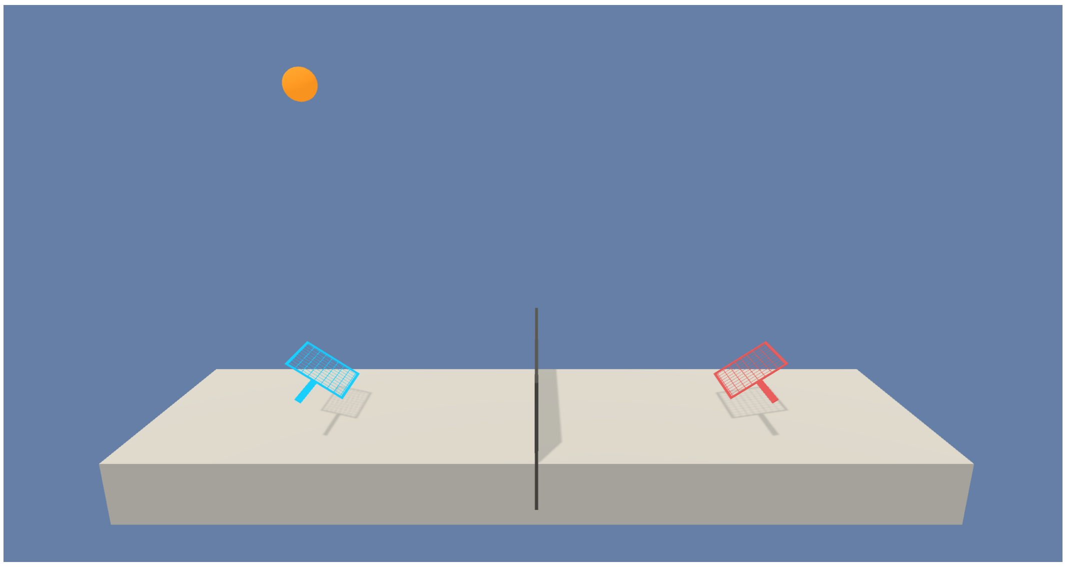
Figure 1: Image of the Unity Reacher environment for the multiple identical agents case [4].

The observation space consists of 8 variables corresponding to the position and velocity of the ball and racket. The length of the state vector is 24, 3 Cartesian directions for each variable. 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, the agents must get an average score of +0.5 (over 100 consecutive episodes, after taking the maximum over both agents). 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 [3].

2 Methodology

In the Unity Tennis environment, both the state and action space are continuous, suggesting that deep deterministic policy gradient (DDPG) [5,6] would have a decent chance of success at training the agents jointly. For this project, a DDPG agent was used. The hyperparameters used in the training of the DDPG agent are listed in table 1. Additional details on the DDPG algorithm, and specifically the implementation used in this project, can be found in the document titled "project2_continuous_control_report.pdf" within the Github repository cited in reference [7].

Despite this being a multi agent environment, the standard DDPG algorithm works to train both the agents because they work cooperatively with one another and they share common goal of keeping the ball in play. They are not actually competing with each other and the game is essentially the same for both agents. Both agents are working together to maximize the total score, meaning they can use the same parameters, i.e. the same actor and critic networks. For other types of multi agent environments, specifically ones in which agents (possibly teams of agents) compete with one another, they can have very different goals meaning that separate policy networks would need to be learned. An example would be OpenAI's work which used deep reinforcement learning to allow agents to learn to play hide-and-seek [8]. The goal of the blue team was to hide from and evade the red team while the red team tried to better learn to find the blue team agents. The tasks of hiding and seeking are dissimilar enough that different policy networks would likely be needed for the agents to perform their tasks properly. However, for the Unity Tennis environment, the goal for the agents was simply to keep the ball in play. The task was identical for each agent, and they both benefited from the other performing well.

Table 1: Parameters used in the DDPG algorithm.

Parameter Name	Value	variable type	description
replay buffer size	$1\text{x}10^5$	integer	maximum number of transitions stored in replay buffer
batch size	128	integer	size of mini-batch sampled from replay buffer used during training
γ	0.99	float	discount factor for future rewards
τ	0.001	float	soft-update mixing parameter for target networks
α_{actor}	$1\text{x}10^{-4}$	float	learning rate for the local actor network
α_{critic}	$1\text{x}10^{-4}$	float	learning rate for the local critic network
number of episodes	4000	integer	maximum number of episodes used for training
$\varepsilon_{\text{init}}$	0.1	float	initial value of the noise decay control parameter
$\varepsilon_{\text{final}}$	0.0005	float	final value of the noise decay control parameter
$\varepsilon_{\text{decay}}$	0.997	float	decay rate of the noise decay control parameter
μ_{OU}	0.0	float	drift parameter for the Ornstein-Uhlenbeck noise [9]
θ_{OU}	0.15	float	growth-rate or decay-rate for the Ornstein-Uhlenbeck noise
σ_{OU}	0.2	float	diffusion coefficient for the Ornstein-Uhlenbeck noise

A parameter, ε, was used to control the amount of exploration done by the agent. When choosing an action, Ornstein-Uhlenbeck (OU) noise was added to the action policy with a probability equal to ε. A random number between 0 and 1 was chosen, and if the random number was less than ε, OU noise was used. The parameters $\varepsilon_{\text{init}}$, $\varepsilon_{\text{final}}$, and $\varepsilon_{\text{decay}}$ from table 1 determine the rate of change of the ε parameter. The ε parameter is decayed to make the actions more deterministic, once the agent has learned a good action policy. However, ε does not immediately start to decay from the first episode. The agent is trained using $\varepsilon = 0.1$ until it solves the environment, and only then does ε start to decay. The $\varepsilon_{\text{decay}}$ parameter sets the scale for this decay and $\varepsilon_{\text{final}}$ is the minimum value ε can take. The difference between $\varepsilon_{\text{init}} = 0.1$ and $\varepsilon_{\text{final}} = 0.0005$ means that until the environment is considered "solved", there is a 10% chance OU noise is injected into the action policy and once ε is completely decayed, this probability drops to only 0.05%. This means that near the end of training, the agent is acting almost entirely according to the deterministic action policy.

3 Results

The collective scores of the two agents are shown in fig. 2. since the returns are so noisy, it's useful to follow the running average (solid orange curve) when assessing the performance of the agents as a function of episode number. It's apparent from the figure that initially the agents are not able to keep the ball in play at all. The average score is essentially zero until around episode 300, except for some initial learning at the beginning that was quickly washed out of the model weights. Even after episode 300, the agents collective average score remains low dipping back down to basically zero several times. In general, the learning starts off very slow and then increases rapidly once the agents randomly stumble upon rewarding actions. Around episode 1250, the agents' score sharply increases and reaches a maximum of about 2.5 at episode 1360. The mean score (dashed orange line) was found by taking the average over the last 400 episodes. The mean value was found to be 1.38 with a variance of 1.144 (standard deviation of 1.070). The fact that the variance is of the same order of magnitude as the mean score indicates a large amount of fluctuation in the scores, which is also evident in the plot of the raw data (solid blue curve). The returns are extremely noisy and the policy very unstable.

The agents first achieve a score greater than +0.5 at episode 1294. Given that the environment is not considered solved until the agents have achieved an average collective score of +0.5 for more than 100 episodes, the environment was first solved after 1394 episodes. The agents maintain this average score of at least +0.5 until episode 1428 when the performance of the agent suddenly deteriorates. For a number of subsequent episodes, the agents continue to perform poorly until again at episode 1512 they achieve an average score of greater than +0.5 meaning the environment is again "solved" at episode 1612. This time the agents are able to maintain a score of $\geq +0.5$ until episode 1849, meaning they were performing satisfactorily for 337 episodes before the performance again depreciated.

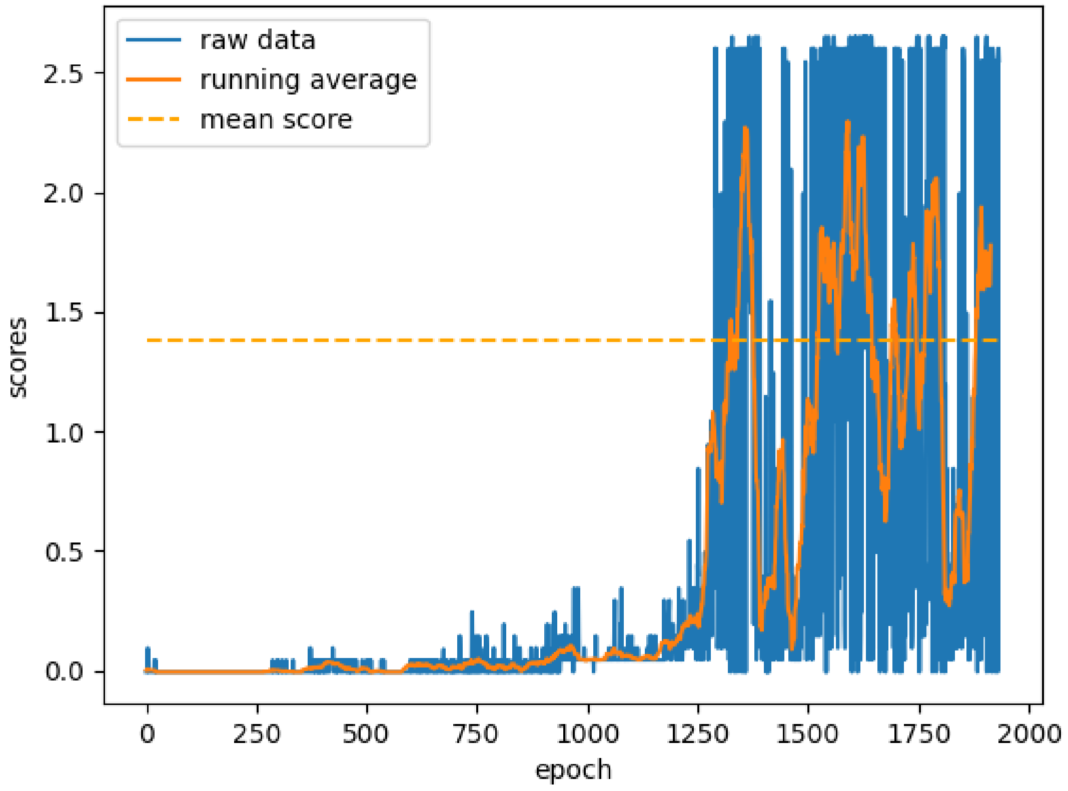

Figure 2: Collective scores for DDPG agents. The solid orange line shows the running average taken over the previous 20 episodes. The dashed orange line represents the mean score taken by averaging from 400 episodes until the end of the training.

4 Conclusion & Future Work

The use of deep neural networks as function approximators to represent state-action value functions, action policies, etc. can in general be unstable [10]. This is further compounded by the fact that this is a multi agent environment; when one agent interacts with the environment it changes the environment for the other agent. This is known as non-stationarity of the environment and, strictly speaking, creates a partially observable Markov decision process [11, 12]. These factors likely contribute to the instability experienced while training the agents. One way to possibly alleviate some of these effects and improve the stability of the algorithm would be to use the weight averaging technique outlined in reference [13].

Even though DDPG works for this multi agent environment, given that the agents are cooperating to achieve a common goal, the use of true multi agent reinforcement learning (MARL) algorithms could still prove beneficial by helping to stabilize learning resulting in a better policy. One such approach is the direct multi agent analog of DDPG called multi agent deep deterministic policy gradient (MADDPG) [14]. The MADDPG algorithm uses a centralized training with decentralized execution framework. This means that during training, all information about the states and actions for all the agents is given to the local critic network, while the local actor networks for each agent only receives local information about their observation space and their own actions. This is a valid approach since during execution, once the agents have been trained, only the actor networks are present so the agents don't receive any extra information which that would not have at their disposal in a real-world scenario. For the case of the Unity Tennis environment, the policy networks would receive information about both agents while the actor networks would only get information related to the individual agents. Since in applying single agent DDPG to this problem, the critic network already has information about both agents, the main difference is that the individual actor policies offered by MADDPG could allow the agents to learn more specialized policies unique to their perspective. However, given that the combined actor network in single agent DDPG is able to generalize over any dimensions that the agents do not experience in the exact same way, the benefits of using MADDPG could be minor.

Other possible multi agent reinforcement learning algorithms which could be applied to this problem are multi agent proximal policy optimization (MAPPO) [15] and Coordinated Proximal Policy Optimization (CoPPO) [16]. These adapt the single agent proximal policy optimization algorithm to the multi agent setting, allowing the use of an on-policy algorithm to solve multi agent environments. These could possibly perform better than the off-policy DDPG algorithm. The authors of the CoPPO paper have proven the monotonicity of policy improvement when optimizing a theoretically-grounded joint objective using their method. This is the same improvement provided by trust region policy optimization [17].

References

[1] Wikipedia contributors, "Multi-agent reinforcement learning — Wikipedia, the free encyclopedia," 2022. [Online; accessed 16-December-2022].

[2] S. S. Selvi, D. K. M V, L. N V, M. P. V. Bhardwaj, and K. S. Shetty, "Deep reinforcement learning algorithms for multi-agent systems - a solution for modeling epidemics," in *2021 IEEE Mysore Sub Section International Conference (MysuruCon)*, pp. 322–327, 2021.

[3] A. Trachte, "Project 3: Collaboration and competition." https://github.com/udacity/deep-reinforcement-learning/tree/master/p3_collab-compet, 2019.

[4] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, and D. Lange, "Unity: A general platform for intelligent agents," *arXiv preprint arXiv:1809.02627*, 2020.

[5] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015.

[6] H. Dong, Z. Ding, S. Zhang, H. Yuan, H. Zhang, J. Zhang, Y. Huang, T. Yu, H. Zhang, and R. Huang, *Deep Reinforcement Learning: Fundamentals, Research, and Applications*. Springer Nature, 2020. http://www.deepreinforcementlearningbook.org.

[7] R. Herchig, "Udacity Continuous Control Project." https://github.com/tauon1777/Udacity-Continuous-Control-Project, 2022.

[8] E. Strickland, "Ai agents play hide-and-seek: An openai project demonstrated "emergent behavior" by ai players - [news]," *IEEE Spectrum*, vol. 56, no. 11, pp. 6–7, 2019.

[9] Wikipedia contributors, "Ornstein–uhlenbeck process — Wikipedia, the free encyclopedia," 2022. [Online; accessed 20-December-2022].

[10] K. Ota, D. K. Jha, and A. Kanezaki, “Training larger networks for deep reinforcement learning,” 2021.

[11] Wikipedia contributors, “Partially observable markov decision process — Wikipedia, the free encyclopedia,” 2022. [Online; accessed 20-December-2022].

[12] H. Kurniawati, “Partially observable markov decision processes (pomdps) and robotics,” 2021.

[13] E. Nikishin, P. Izmailov, B. Athiwaratkun, D. Podoprikhin, T. Garipov, P. Shvechikov, D. P. Vetrov, and A. G. Wilson, “Improving stability in deep reinforcement learning with weight averaging,” 2018.

[14] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, “Multi-agent actor-critic for mixed cooperative-competitive environments,” 2017.

[15] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, and Y. Wu, “The surprising effectiveness of ppo in cooperative, multi-agent games,” 2021.

[16] Z. Wu, C. Yu, D. Ye, J. Zhang, H. Piao, and H. H. Zhuo, “Coordinated proximal policy optimization,” 2021.

[17] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, “Trust region policy optimization,” 2015.