Udacity – Deep Reinforcement Learning Nanodegree

Collaboration and Competition project

Deep reinforcement learning

Marwa Kandil
11-8-2021

Table of Content

1. Introduction

Reinforcement learning (RL) is one of the most fascinating concepts of AI as machines learn from their own experiences and optimize their actions based on experience. The agents are rewarded or penalized based on their actions and the actions that reach the optimal outcome are rewarded. Agents of RL managed to learn actions in complex dynamic environments but are limited due to intensive computations. Recently, RL is combined with deep neural network resulting deep Q-network (DQN) surpassed all previous algorithms and was able to learn concepts such as object categories directly from raw sensory data [1]. Deep reinforcement learning (DRL) agents, such as DQN, can solve high dimensional observational spaces such as achieving human score in Atari games and chess. However, DQN are limited as it cannot solve physical control tasks as tasks with continuous and high dimensional action spaces. To solve continuous control problems, actor-critic algorithms were designed to utilize value-based and policy-based RL methods together to train the agents.

In real world, most tasks require interaction between multiple agents to solve a particular problem, such as delivery drones. This gave rise to Multi-agent Reinforcement Learning (MARL) algorithms where several agents interact together to better understand the environment and optimize action selection. Some benefits of MARL algorithms include sharing of knowledge between agents making them smarter, and a more robust system where failed agents can be replaced. Markov Game is a tuple that consist of set of all actions from all agents, and it is used to solve MARL. The way rewards systems are defined identifies agents type, such as collaborative or competitive or mixture of both. Recently, actor-critic and MARL algorithms were combined to solve multi-agent problems, by utilizing multiple-agents with DDPG. In MADDPG, actors have access to local information while critics can learn from extra information about the policies of other agents.

There are many open-source tools train Deep reinforcement learning (DRL) agents to encourage large-scale innovation for multiple fields such as video games. Unity Machine Learning Agents (ML-Agents) is one of the main open-source tools that provide intelligent agents and rich environment for game development. The ML-agents Train and embed intelligent agents by leveraging state-of-the-art deep learning technology [2]. In this project I used Unity's rich environments to design, train, and evaluate actor-critic DRL algorithms.

For this project, I trained two agents control rackets to bounce a ball over a net. The environment has state space of 8 variables corresponding to position, rotation, velocity, and angular velocities of the arm. The agent generates an action vector with four numbers, corresponding to the position and velocity of the ball and racket. Each agent receives its own, local observation. The agent gains a reward of +0.1 for if it hits the ball over the net. If an agent lets a ball hit the ground or hits the ball out of bounds, it receives a reward of -0.01. The aim of each agent is to keep the ball in play. The task is solved when the average score is +0.5 for 100 consecutive episodes. I used two algorithms to solve this problem. The first algorithm used was vanilla Multi-agent Deep Deterministic Policy Gradient (MADDPG) and the second algorithm used is a MADDPG with prioritized replay buffer.

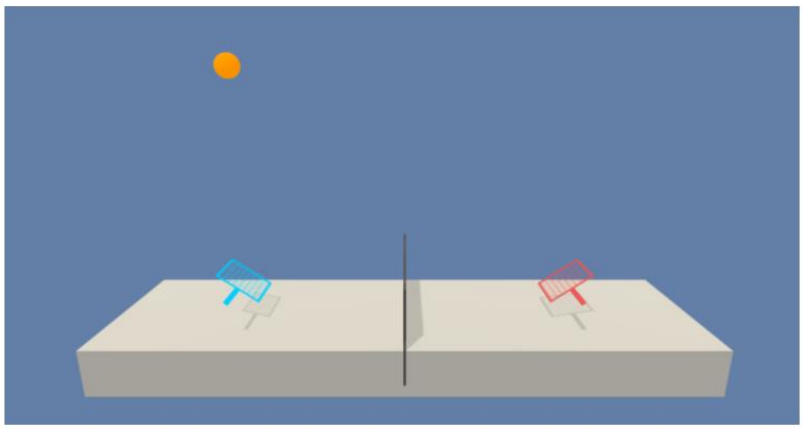

Figure 1: Unity ML-agent Tennis Environment

2. Background

Reinforcement learning framework consists of five main parts: states, actions, environment, agent, and rewards [3]. States are the different situations returned by the environment. Actions are set of possible moves that the agent can take to optimize the reward. The agent is a software that utilizes a policy to determine the next action based on the current state. The agent's role is to evaluate different policies and identify the policy that maximizes reward. Thus, the agent evaluates the expected long-term return of an action taken under policy at the current state also known as the action-value function Q(S,A). The action-value function uses discounted rewards as the effect of the reward is reduced the further away we move in the future. The RL agent uses Bellman equation as an iterative update to estimate the action-value function such that:

$$Q^*(s,a) = E_{s'}[r + \gamma max_{a'} Q^*(s',a')|s,a]$$

Where the variables s and a stand for state and action, Q^*(s,a) stands for action-value function for s and a, γ stands for reward discount rate. Non-linear function approximators of neural networks can be used to estimate the action-value function, such as Deep Q-network (DQN) [1]. Actor-critic (AC) algorithms aim to use the value-based techniques to further reduce the variance in policy-based methods. Value based methods, such as DQN, uses NN to optimize the action-value function. Policy-based methods use NN to approximate policy functions, thus, optimizing the action-value function but suffers from high variance. AC agents learn from their actions and uses the critic to identify good and bad actions. Thus, AC are more stable than value-based, and need less samples and time than policy-based. Deep Deterministic Policy Gradient (DDPG) is an actor-critic method where the critic uses a maximizer to approximate the Q-value over the next state [4]. The actor estimates the optimal policy deterministically, by generating the best possible action at any input state. The critic is updated using the loss function between the expected Q-value and the target Q-value. the actor policy is updated using the sampled policy gradient as the equation below. DDPG uses replay buffers and preforms soft-update to the target network with ratio of the regular network to allow faster update.

$$\nabla J_{\theta\mu} \approx \frac{1}{N} \sum_i \nabla Q(s,a|\theta^q)\Big|_{s=si,\, a=\mu(si)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{si}$$

To adopt DDPG with multi-agent (MADDPG) setting the framework will use centralized training with decentralized execution [5]. This will allow the policies to use extra information from other agents to ease training, however, this information is not used at test time. The critic uses centralized training where it uses extra information about the policies of other agents. After training, the actor is executed using local information in a decentralized manner and equally applicable in cooperative and competitive settings.

Similar to DQN, MADDPG uses experience replay buffer as it allows agents to learn more from individual tuples multiple times, recall rare occurrences, and in general make better use of experience. The actor and critic are updated at each timestep by sampling a minibatch uniformly from the buffer. One possible way to improve MADDPG implementation is to use prioritized experience replay to improve learning. Prioritized experienced replay is based on the idea that the agent can learn more effectively from some transitions than from others, and the more important transitions should be sampled with higher probability [6].

3. Implementation and Results

In this paper I implemented vanilla MADDPG and MADDPG with prioritized experience replay (MADDPG_PER) to solve the 2 agent Tennis problem. The neural network model that I used for the critic consists of one input layer that has same shape as the number of states + actions, one linear hidden layer with number of inputs as 256 and number of outputs as 128. For training, Adam optimizer was used with learning rate of 1e-4. The neural network model that I used for the actor consists of one input layer that has same shape as the number of states, one linear hidden layer with number of inputs and number of outputs as 256 and 128. For training, Adam optimizer was used with learning rate of 1e-3.

For both implementations, MADDPG and MADDPG_PER, I used a replay buffer of size 100,000 unit and 250 mini batch size, discounted factor (ɣ) of value 0.99, and soft-update ratio (τ) of 1e-3. To ensure balanced trade off between exploration and exploitation, Ornstein-Uhlenbeck noise generator for exploration, noise decay rate of 0.99. The scores throughout the episodes were tracked and presented on the plot below every 100 episodes. As shown the MADDPG model solved the problem in 1880 episodes and the MADDPG_PER solved the model in 3193 episodes. The MADDPG model was much faster to reach the solution, but the Figure below shows clearly that there was a sudden jump in the score values. This indicates that the model is not very stable. On the other hand, MADDPG_PER too longer to solve the problem, yet the learning trend was much smother. When using prioritized experience replay, initially the sampled experiences are random but as time progresses experiences with higher importance have higher priority. Therefore, learning with MADDPG_PER was much more stable than with MADDPG.

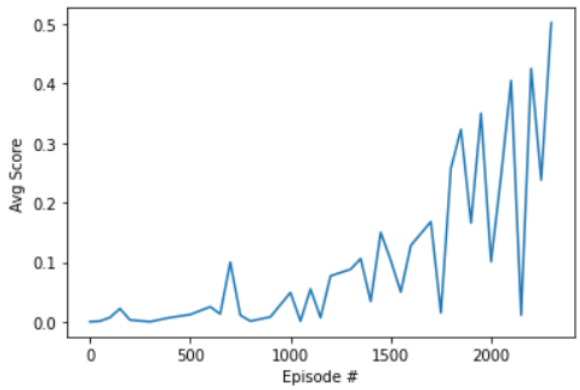

Figure 2: MADDPG average score vs training episodes

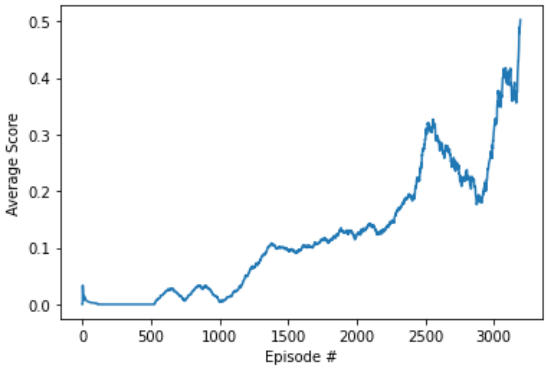

Figure 3: MADDPG_PER average score vs training episodes

Algorithm	Total number of episodes	Average Score	Test score
MADDPG	1880	0.502	2.38
MADDPG_PER	3193	0.503	2.75

4. Conclusion and Future Work

In this project, I implemented MADDPG and MADDPG with priority experience replay to train and evaluate two agents control rackets to bounce a ball over a net in the unity ML-agent Tennis environment. The results show that using MADDPG the agent solved the problem in 1880 episodes, on the other hand, MADDPG_PER solved the problem in 3193 episodes but the leaning process was more stable.

Future work to the current solution would be to experiment with Proximal Multi-Agent Policy Optimization (PPO) to determine the appropriate policy with gradient methods. Furthermore, the solution can be modified to solve the environment by implementing A3C method where multiple agents who all interact with the environment separately and send their gradients to the global network for optimization in an asynchronous way.

5. References

[1]: V. Mnih et al., "Human-level control through deep reinforcement learning", Nature, vol. 518, no. 7540, pp. 529-533, 2015. Available: 10.1038/nature14236 [Accessed 3 September 2021].

[2]: U. Technologies, "Machine Learning Agents | Unity", Unity, 2021. [Online]. Available: https://unity.com/products/machine-learning-agents. [Accessed: 03- Sep- 2021].

[3]: R. Sutton and A. Barto, Reinforcement Learning, 2nd ed. 2019.

[4]: T. P. Lillicrap et al., "Continuous control with deep reinforcement learning", arXiv, vol. 150902971, 2015. [Accessed 4 October 2021].

[5]: R. Lowe et al. , "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments", arXiv, vol. 170602275, 2017. [Accessed 24 October 2021].

[6]: T. Schaul, J. Quan, I. Antonoglou and D. Silver, "Prioritized Experience Replay", arXiv, vol. 151105952, 2016. [Accessed 4 October 2021].