Tennis matchup agent based on MADDPG

Project Objectives:

Train two agents who play tennis so that they can catch the ball as they would in a tennis match. The goal of both agents is not to let the ball fall to the ground, to allow the ball to bounce between the rackets at the same time, not to fall or cross the border, and then to maintain the ball in the game.

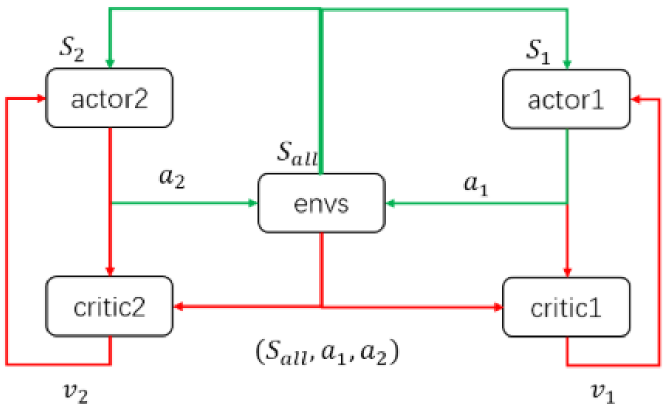
The property of the environment is a similar but different environment to the Tennis environment on the Unity ML-Agents GitHub page. In the environment, the racket is controlled by two agents to play on the net. If an agent hits the ball with the net, it will receive a reward of +0.1. If an agent makes a ball hit the ground or knock it out of bounds, its reward is -0.01. Therefore, the goal of each agent is to keep the ball in the game. The observation space consists of 8 variables, which correspond to the position and speed of the ball and racket, respectively. Each agent receives its own local observations. There are two continuous actions available, which correspond to moving towards (or away from) the network and jumping. The task is episodic, and in order to solve the environment, the agent must obtain an average score of +0.5 (after both agents have achieved the maximum score, 100 consecutive times). At the same time, without discount, the rewards obtained by each agent are added up after each episode to obtain its score, and the largest one of the 2 scores generated is used as the score to obtain a single score for each episode.. When the average of these scores (over 100 episodes) is at least +0.5, the environment is considered resolved.

Project Flow:

Reinforcement learning related projects first need to clarify the three elements of the state of the environment, actions and rewards. As can be seen from the project objectives above, the observation space of the agent is composed of 8 variables, which correspond to the position and speed of the ball and racket, respectively. Each agent will receive its own environmental observation data, and there are two consecutive actions available, corresponding to the movement and jump towards (or away from) the network, so the environment state of a single agent consists of 12 values (8 +4), because this is a multi-agent environment, the state of another agent needs to be considered, so it is 24 values. Because the agent must obtain an average score of +0.5, it is necessary for both agents to obtain the highest score for more than 100 consecutive times.

The environmental state of multi-agents is very different from the traditional environment. It is usually determined by the behavior of multiple agents. It is inherently unstable, and traditional reinforcement learning algorithms are difficult to train and converge. In this project, two agents need to be used to achieve the goal, so I chose to use the MADDPG algorithm explained in Udacity, in order to perform centralized learning and decentralized execution in a multi-agent environment, so that multiple Agents can better learn the cooperative mode, so that they can continuously control the ball. The MADDPG algorithm inherits the AC framework in reinforcement learning and is extended by the DDPG algorithm based on this framework.

Think of each agent in the model as a different "actor", and actors 1, 2 will get advice from "reviewers" to decide which behaviors should be strengthened during training. Traditionally, reviewers will try to predict the future value (reward) of an action in a particular state, and the agent will use the action to update its strategy. This method is much more stable than directly using the rewards of a certain stage, and also interrupts its state continuity through soft updates. In order to enable the model to coordinate the behavior of all agents globally, the MADDPG algorithm enhances the strength of the reviewer so that it can access the states and actions of all agents. The specific process is shown in the figure below.



(Picture from https://www.cnblogs.com/initial-h/p/9429632.html)

When the model is trained, the agent only needs to interact with the environment, as shown in the green circle in the figure.

Code composition:

In this project, my code is mainly composed of four parts, namely: neural network, agent, hyperparameter, training.By referring to the code of MADDPG in Udacity, I use its same model to build neural networks and agents. In the agent code part, there are mainly three parts: behavior function, learning function, and soft update. The main action is to update the weights through the evaluation of the Critic network after the agent takes action through the Actor network.

In the code defined by the agent, there is an important hyperparameter epsilon that controls the agent's exploration. When the value is closer to 1, the more important it is the future total reward. When the value is equal to 0, the agent only Value the rewards of the current state, regardless of the future. The multi-agent algorithm model can use the Epislon-greedy strategy to control the appropriate exploration environment of the agent. This method reduces the probability for exploration with increasing time. This project uses the SGD optimizer to update the learning rate by multiplying the diminishing factors in each period. Here, a method similar to Epsilon Greedy is used to achieve it. Finally, Ornstein-Uhlenbeck process and empirical playback function are used.

To solve the problem of gradient explosion, the model uses the torch.nn. utils.clip_grad_norm function to implement gradient clipping, and sets the range of the gradient to 1. Prevents them from growing exponentially by setting an upper limit on the size of parameter updates. This can make the agent more stable and fast learning.

In order to explore the action space more, the EPS_START parameter is set to 6.0, so the chance of detecting a signal can be increased. When the noise drops to zero, this extra signal seems to improve learning in subsequent training.

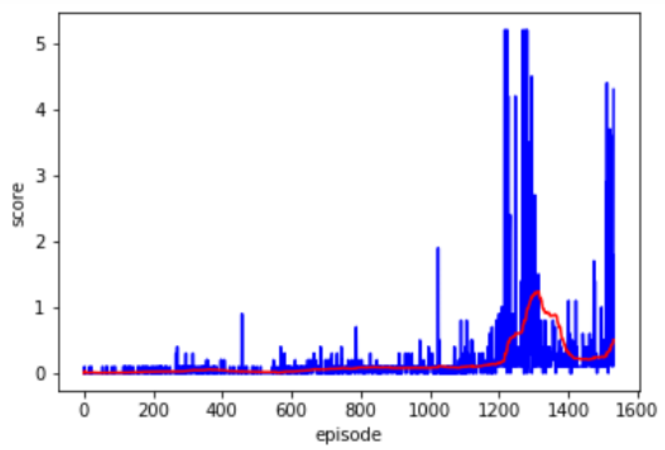
In the training part, by setting the learning interval, multiple learnings per plot are performed to produce faster convergence speed and higher scores. But setting the learning interval, which slows down training, is a question worth weighing. Finally, I set the learning interval to 1, so that the agent will perform the learning steps in each plot, so that it samples experience from the experience pool and runs the learning function of the agent part.

About hyperparameters:

EXPERIENCE_POOL_SIZE = 1000000 # define the experience pool size
BATCH_SIZE = 128 # minibatch size
LR_ACTOR = 0.001 # actor network learning rate
LR_CRITIC = 0.001 # critic network learning rate
WEIGHT_DECAY = 0 # L2 weight decay
LEARN_EVERY = 1 # learning interval
LEARN_NUM = 4 # learning coefficient
GAMMA = 0.99 # discount factor
TAU = 7e-2 # soft update parameters
OU_SIGMA = 0.2 # Ornstein-Uhlenbeck noise parameter, volatility
OU_THETA = 0.15 # Ornstein-Uhlenbeck noise parameter, mean recovery speed
EPS_START = 6.0 # Initial value of ε during noise attenuation
EPS_EPOSIDE_END = 300 # End noise attenuation after a specified number of trainings
EPS_FINAL = 0 # final value of ε after attenuation
GOAL_SCORE = 0.5 # goal score
WINDOW_LENGTH = 100 # limited number of actions
PRINT_EVERY = 10 # print the score every tenth time
ADD_NOISE = True # add noise

Operation result:

The following figure shows the final training results.



After 1600 episodes of training, the agent can solve the problem of the project environment. The highest score is 5.2 and the highest average is 1.219

Future Development:
Algorithmically:

First of all, I think the most important improvement of reinforcement learning is the use of transfer learning. If the agent can continuously improve through memory, then general artificial intelligence is just around the corner. Then, the priority experience playback algorithm is not used in this model training. The priority experience reply does not learn the data stored in the experience pool through random selection, but chooses the experience based on the priority value related to the error size.

At Work:

I will consider using MADDPG for medical diagnosis. By treating the patient's physical state as a dynamic environment and each external drug dose as an action, I will explore the application of artificial intelligence in medicine in this way.