Project Collaboration and Competition

Introduction

In this project 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.

Solution

The algorithmic approach used to solve the problem is an adaptation of Multi-Agent Deep Deterministic Policy Gradients (MADDPG) algorithm as proposed by Lowe, Ryan, et al. "Multi-agent actor-critic for mixed cooperative-competitive environments." for the collaborative self-play agent. We first describe the basics of the DDPG algorithm.

DDPG uses four neural networks: a Q network, a deterministic policy network, a target Q network, and a target policy network. DDPG is an Actor-Critic method and in DDPG, the Actor directly maps states to actions. The target networks are

time-delayed copies of their original networks that slowly track the learned networks. Using these target value networks greatly improve stability in learning.

The algorithm uses the following main components:

1. Experience replay: we save all the experience tuples (state, action, reward, next_state) and store them in a finite-sized cache — a "replay buffer." Then, we sample random mini-batches of experience from the replay buffer when we update the value and policy networks.

2. Actor & Critic network updates: The updated Q value is obtained by the Bellman equation. In DDPG, the next-state Q values are calculated with the target value network and target policy network. Then, we minimize the mean-squared loss between the updated Q value and the original Q value.

3. Target network updates: We make a copy of the target network parameters and have them slowly track those of the learned networks via soft updates

4. Exploration: For continuous action spaces, exploration is done via adding noise to the action itself. We use *Ornstein-Uhlenbeck Process* to add noise to the action output (same as used by the original DDPG authors)

The flowchart for the DDPG algorithm [Lillicrap et al, 2015] is given below:

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for

In our implementation, the main difference from MADDPG is that agents share actor and critic models. The actor model learns to predict an action vector given the state of the environment as observed by the agent. The critic model learns Q-values for combined states and actions from all the agents in the environment. In this way, the actor only relies on local information while the critic uses global information.

Network Architectures

The Actor network maps states to action. It has an input dimension of 24 units, two fully connected layers with 128 and 128 units with relu activation, batch normalization and tanh activation for the action space.

The Critic network maps combined states and actions of both agents to Q values. It has an input dimension of 48 (24x2), utilizes two fully connected layers with 128 and 128 units with relu activation and batch normalization. In put for actions for both agents (2x2 = 4) is inserted between the first and second fully connected layers.

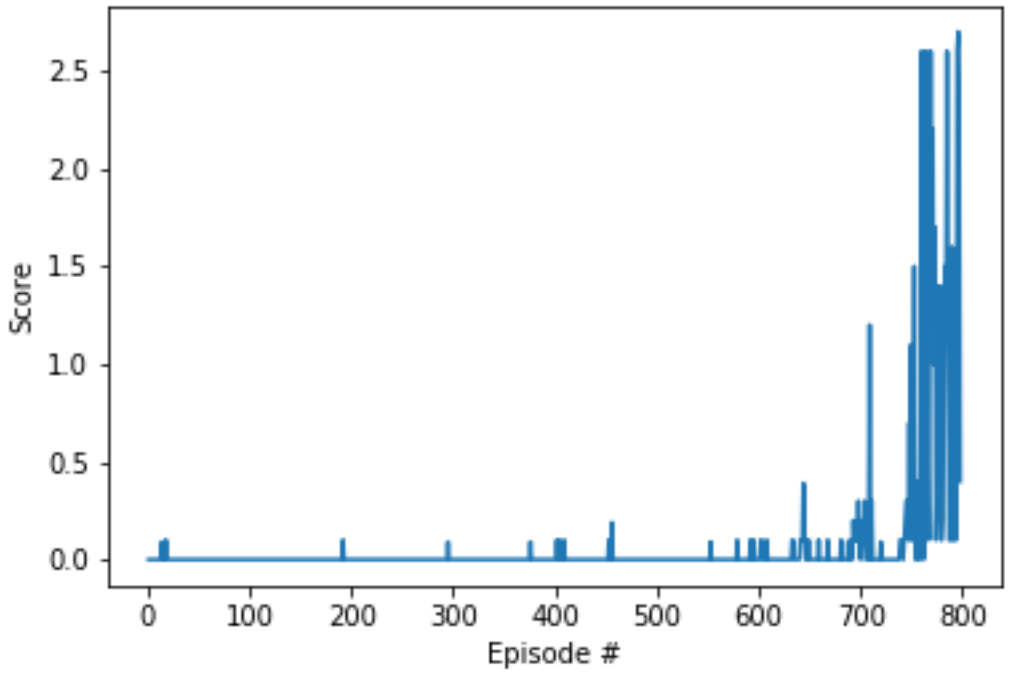
Hyperparameters

Name	Description	Value
buffer_size	replay buffer size	1e6
batch_size	mini batch size	128
discount	discount_factor	0.99
target_mix	factor to weight nw weights	3e-3
lr_actor	learning rate for actor	1e-3
lr_critic	learning rate for critic	1e-3
noise_mu	Mean for noise process	0
noise_theta	Theta for noise process	0.15
noise_sigma	Sigma for noise process	0.1
actor_fc1	Number of neurons in first fully connected layer for the actor	128
actor_fc2	Number of neurons in second fully connected layer for the actor	128
critic_fc1	Number of neurons in first fully connected layer for the critic	128

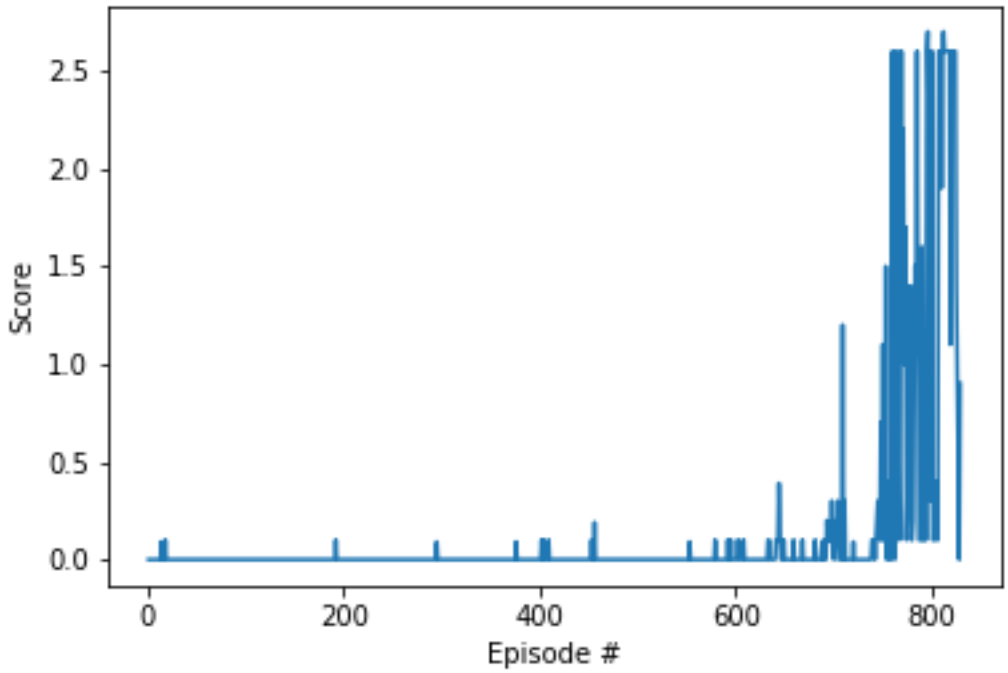
critic_fc2	Number of neurons in second fully connected layer for the critic	128
max_episodes	Maximum number of episodes in training	2000
max_steps	Maximum steps in each episode	1e6

Results

The figure below shows the performance of the algorithm to solve the environment. We achieved average score of 0.50 from iteration 698.



If the target score was increased to average of 1.0 over 100 episodes, the agents were able to reach that performance in 729 episodes as shown below. This means the agents rapidly improved in the last episodes.



Future Work

Explore an actor model that learns actions for both agents at the same time, for example input are observations and output are actions of both agents.