Report

The implementation of this reacher (single/multi-agent) solution was done in a python 3.6, with GPU (cuda) support in a local (linux ubuntu 24.04) environment.

Learning Algorithm

The learning algorithm was an implementation of actor-critic DDPG method, with various combinations of hyperparameter and model structure attempted to achieve the solution.
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. Every entry in the action vector should be a number between -1 and 1.

How it works:
This is a multi agent model, where each agent (2, representing 2 players) has a pair of Actor neural nets (actor_local, actor_target) with identical structure, and another pair of Critic neural nets (critic_local, critic_target) with identical structure. There is one replay buffer shared across all the agents.

Actor neural network:

> 24 inputs for each dimension → 256 unit hidden layer → ReLu activation → 128 unit hidden layer → ReLu activation → 4 unit output for each action → tanh activation (to bound it between -1 and +1)

Critic neural network:

> 24 inputs for each dimension → 256 unit hidden layer → Batch Normalization → ReLu activation + action (concatenated) → 128 unit hidden layer → ReLu activation → 1 unit output

In every step of iteration, actor_local from each agent produces action (with added noise, following the Ornstein-Uhlenbeck process and parameters: mu=0., theta=0.15, sigma=0.08) for each action types, and trains itself. With these actions fed back into the environment, which returns the next states corresponding to each action for each agent, the whole individual experiences (state, action, reward, next_state, done for all agents combined) from individual agents get added to the shared replay buffer to facilitate cross learning.

The algorithm use a fixed buffer of 1,000,000 size, and was trained upto 5,000 episode with each episode having 1,000 steps. The model was trained in every steps, and the experience from each step was added to the fixed buffer (which deleted oldest experience when full to accommodate for newest experiences). For each step of learning, 512 experiences were sampled randomly from the replay buffer, which captures state, action, reward, next_state, done (for all agents combined to facilitate cross-learning).

In each training iteration, each agent learns by updating the critic, by getting predicted next state actions from actor_target, and then getting the Q values for next states by feeding those next state and next state actions into the critic_target network. The Q targets for current states are calculated as follows: rewards + (gamma * Q values for next states * (1 – dones)). In other words, Q targets for current states = rewards + gamma * critic_target(next_state, actor_target(next_state))
Q expected is calculated using current state and action from critic_local. Using Q target and Q expected, losses are calculated, which is then used for training the critic_local network through back-propagation of the losses.

Then actor in each agent is updated, by back-propagating the actor losses (derived by putting current states in actor_local to get predicted actions and then feeding these predicted actions and current states into critic_local) through the actor_local.

Once the local networks (critic_local and actor_local) are updated, then it is followed by soft update of the target network parameters (for both critic and actor with parameter TAU (target = TAU * local + (1 - TAU)*target)

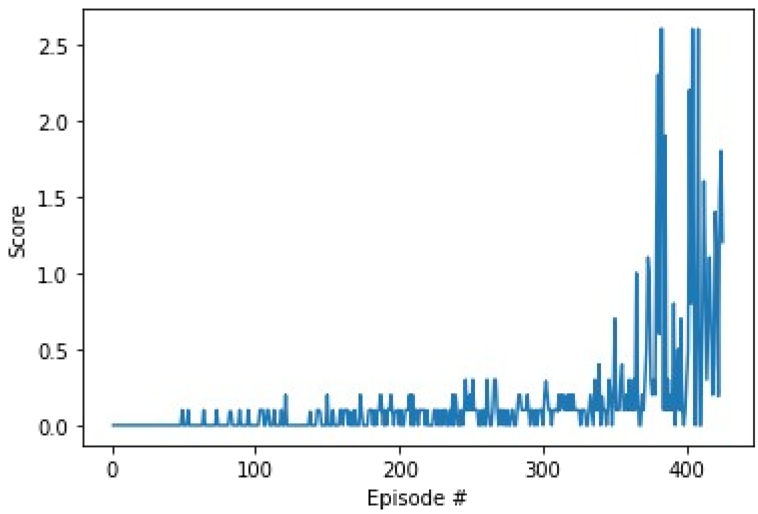
Other hyperparameters

    BUFFER_SIZE = int(1e6) # replay buffer size
    BATCH_SIZE = 512   # minibatch size
    GAMMA = 0.99   # discount factor
    TAU = 0.01   # for soft update of target parameters
    LR_ACTOR = 1e-4   # learning rate of the actor
    LR_CRITIC = 1e-3   # learning rate of the critic
    WEIGHT_DECAY = 0   # L2 weight decay
    NUM_EPISODES = 5000
    MAX_T = 1000
    PRINT_EVERY = 100

Various hyperparameters were tried by changing the hyperparameters, but best results (below) were achieved with the hyperparameter values above.

Plot of Rewards

```
Episode 100     Average Score: 0.01
Episode 200     Average Score: 0.04
Episode 300     Average Score: 0.08
Episode 400     Average Score: 0.30
Episode 425     Average Score: 0.51
Environment solved in 425 episodes!     Average Score: 0.51
```



Ideas for Future Work

Approach similar to AlphaZero can be used. Additionally, PPO or D4PG can also be tried. Also play with neutral networks structure and hyperparameters to achieve better results.