Report Tennis

Author: Dorzhi M.

Approach description

We apply DDPG to solve a multi-agent problem, where two agents control rackets to bounce a ball over a net. The approach we apply is based on 'Continuous control with deep reinforcement learning' (Lillicrap et al.). We adapt code from project 2 for multiple agent setting.

This problem is a reinforcement learning problem with two agents. We have two agents interacting with each other, and each agent receives rewards after each action.

Reinforcement learning problems can be addressed using value based and policy based methods. Unlike DQN, where we try to learn a value function of the action state space, policy based methods learn directly from the observation space by mapping states to actions using the gradient ascent.

One limitation of DQN is that it works only for discrete action space. DDPG combines actor-critic approach and the insights from DQN for tackling continuous problems. In DDPG we have two neural networks, one neural network, the 'Actor', is used to approximate the optimal policy deterministically, while the other network, the 'Critic', learns to evaluate the action value using Bellman equation as in DQN.

As DQN, DDPG utilizes a replay buffer to reduce bias from correlation between sequential samples. What is more, in DDPG, unlike DQN, target networks (both actor and critic) are updated using a "soft" update, which slowly mixes-in the regular network weights to the target network weights.

The observation space consists of 8 variables corresponding to position and velocity of the ball, and each agent receives a local version of the observation. Each action is a vector with two numbers, corresponding to movement toward (or away from) the net, and jumping. 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. 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.

Both actor and critic are fully connected neural network with 2 hidden layers. The following neural net is used for parametrizing Q function:

- 8-dimension input layer
- 1st hidden layer with dimension of 400, 2nd hidden layer with 300 dimensions

- 2-dimension output layer

In order to solve the environment, the agent must get an average score of +0.5 over 100 consecutive episodes.

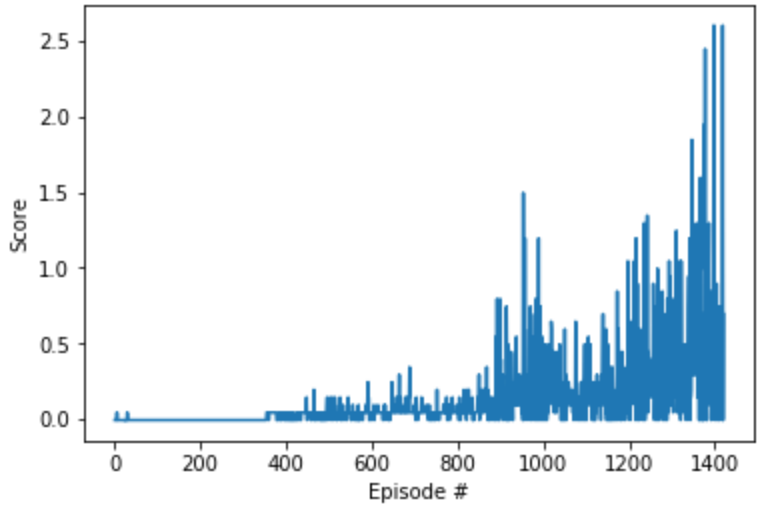
We set following hyperparameters:

- maximum number of training episodes (by default 5000)
- maximum number of timesteps per episode (by default 1500)
- replay buffer size (by default 1e6)
- minibatch size (by default 512)
- discount factor gamma (by default 0.99)
- soft update of target parameters tau (by default 1e-3)
- learning rate of the actor (by default 1e-4)
- learning rate of the critic (by default 1e-3)
- L2 weight decay (by default 0)

At first, all the hyperparameters were taken from the example project. We got pretty bad performance.

Agent Performance

After we increased the replay buffer size to 1e6, we were able to solve the problem in 1420 episodes.



Ideas for future

As next steps, we can improve the performance of the agent by using prioritized experience replay, as in here. Some of the transitions are more important for the agent performance (those with higher error), and we can sample those transitions with higher probability, potentially speeding up the training process.