Collaboration and Competition Project

Troy Chevalier

January 29, 2019

> This project uses the Unity Tennis environment. In this 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.

Learning algorithm

Environment

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. There are three stacked observations; therefore, the observation space consists of 24 variables.

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, 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.

Model architecture

The implementation uses the Multi-Agent Actor-Critic Deep Deterministic Policy Gradient (MADDPG) algorithm [Lowe et al., 2017]. The motivation for the algorithm is that traditional reinforcement learning approaches such as Q-Learning or policy gradient are poorly suited to multi-agent environments. One issue is that each agent?s policy is changing as training progresses, and the environment becomes non-stationary from the perspective of any individual agent. This presents learning stability challenges and prevents the straightforward use of past experience replay, which is critical.

The authors adopted a framework of centralized training with decentralized execution, allowing the policies to use extra information to ease training, so long as this information is not used at test

time. This is achieved by providing the critics with access to the environments of both agents. The actors only have access to local information.

Deep Deterministic Policy Gradient (DDPG) [Silver et al., 2014] is an off-policy actor-critic algorithm using deep function approximators that can learn policies in high-dimensional, continuous action spaces. It adopts several ideas from DQN [Mnih et al., 2013]: 1. the network is trained off-policy with samples from a replay buffer to minimize correlations between samples; 2. the network is trained with a target Q network to give consistent targets during temporal difference backups.

In my implementation there are two agents: both the actor and the critic use a feedforward neural network. Each network has two hidden layers each with 256 parameters. The neural networks used the leaky rectified non-linearity for all hidden layers. The output of the actor uses a *tanh* function to bound the actions between -1 and 1. The actor network architecture is depicted in Fig. 1.

The critic network architecture uses both sets of agent observations and the action associated with that agent.[1] That makes a total of 50 input parameters. There is only one output, the value of being in that state and taking the given action. The critic network architecture is depicted in Fig. 2.

[1] This is different from the paper, where the critic would get the actions from both agents. I elected to simplify the implementation.

Training

The plot of rewards per episode is shown in Fig. 3.

Hyper-parameters

With MADDPG there are many hyper-parameters to tune. These include the following: (1) the architecture (number of hidden layers and number of parameters) of the actor and critic neural networks, (2) τ, the parameter used for Polyak averaging for the target networks, (3) the type of action noise and the rate of decay, and (4) how often to train the networks and the batch size.

The networks used in the DDPG paper [Lillicrap et al., 2015] were particularly complicated. Batch normalization was used for the state input and all layers of the actor. The critic used batch normalization prior to the action input and the actions were not incorporated until the hidden layer. I went with a much simpler network with no batch normalization and the actions incorporated into the input layer of the critic.

For the target networks, I tried various values for τ, from 0.001 to 0.1. I found that 0.005 worked well.

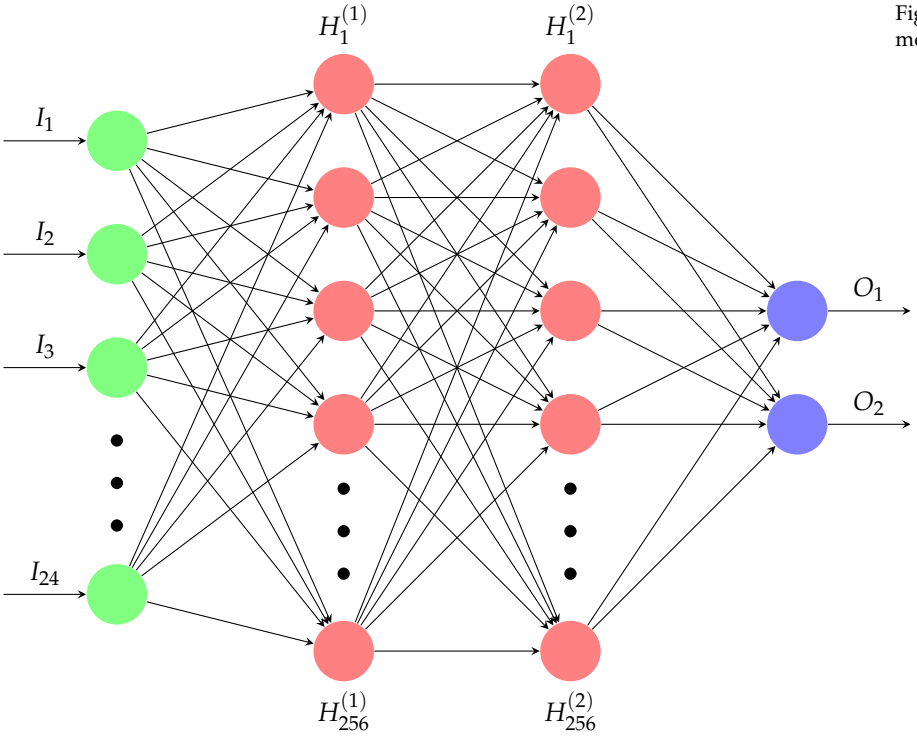

Figure 1: Feedforward network used to model the actor.

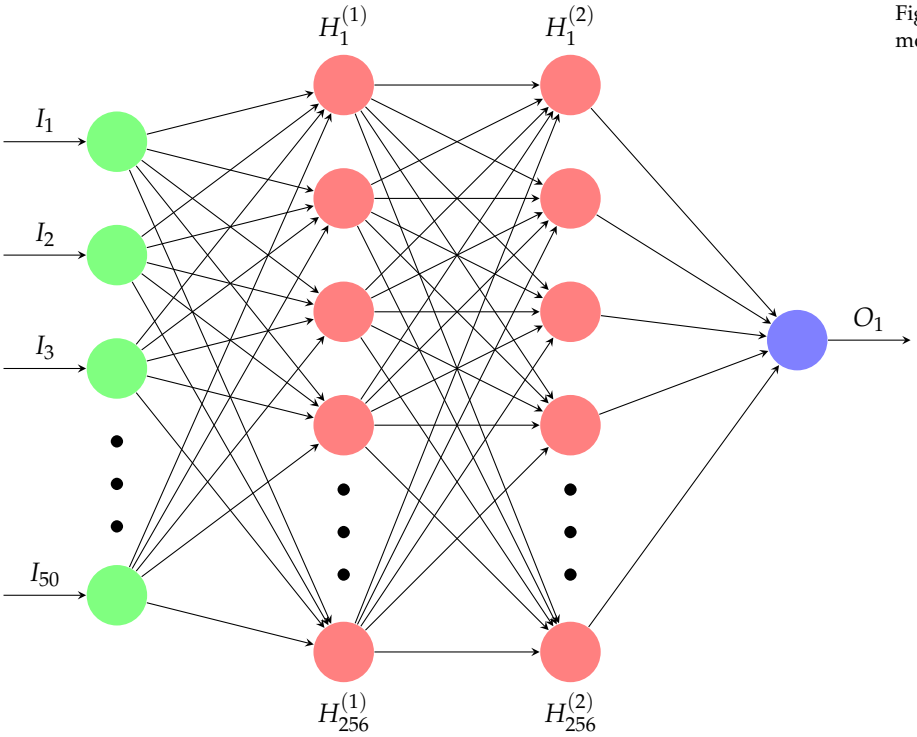

Figure 2: Feedforward network used to model the critic.

The choice of action noise is an important one. I tried using both Ornstein Uhlenbeck (OU) noise and Gaussian noise for the action noise. OU noise worked better for this task. That isn't surprising, because it's desirable if the agents move smoothly and for them to return to their original position. I explored a suggestion from the OpenAI folks, which is to use uniform random noise during the initial *warmup* phase; however, I don't feel that it helped for this task. It is a configurable option in the code.

One of the most important decisions is how often to train the networks. In the DDPG and MADDPG papers, the networks were trained every 100 steps. Given the short episodes and the sparseness of the rewards, this was not often enough. I ended up adopting the approach used in the Soft Actor Critic paper and did a training update every step. I used a batch size of 128; I didn't find the choice of batch size to be as critical.

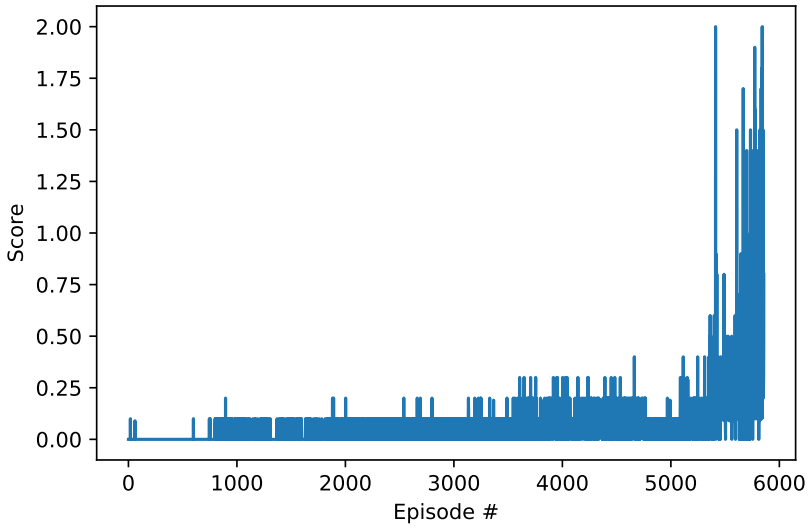

Figure 3: Rewards per episode.

Future work

There are two main areas that I would like to pursue. First, reduce the amount of initial OU noise and see if the agents can move less. They are cooperating agents, so ideally they shouldn't move unnecessarily.

Second, DDPG and MADDPG are brittle algorithms that are difficult to tune. I implemented the Soft Actor Critic algorithm [Haarnoja et al., 2018], but I haven't had enough time to train it to solve the environment.

References

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pages 6379–6390, 2017.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.