12 July 2019 10:59

Collaboration and Competition

Learning Algorithm

In this project, 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.

I chose to use the Deep Deterministic Policy Gradient algorithm, similar to the one used previously within the Continuous Control project, using two copies of the same agent.

The standard DQN is not ideal for a continuous action space, and while adapting it for a continuous space would imply discretizing the action space, where the number of actions increases exponentially with the number of degrees of freedom. Such large action spaces are difficult to explore efficiently, and thus successfully training DQN-like networks in this context is likely intractable. Additionally, naive discretization of action spaces needlessly throws away information about the structure of the action domain, which may be essential for solving many problems. Instead I used a model-free, off-policy actor-critic algorithm using deep function approximators that can learn policies in high-dimensional, continuous action spaces, called DDPG (based on paper)

In DDPG, there are two networks, one for the actor and one for the critic. The actor is used to approximate the optimal policy deterministically, generating the best belief (argmax Q(s,a)). The critic learns to evaluate optimal action value function by using the actor's best belief. In fact the actor is an approximate maximiser which calculates a new target value for training the action value function.

The original DQN has two copies of the network weights (Regular/Target). The weights of the regular network are copied into the target network every 10000 steps. Within DDPG, there are two copies of the network weights for both the actor and the critic. The target networks are updated using a soft update strategy, by slowly blending the regular network weights with the target network weights. Basically every step there is a mix in 0.01% of regular network weights with the target network weights, which facilitates faster convergence.

Hypermeters:

```
BUFFER_SIZE = int(1e6)   # replay buffer size
BATCH_SIZE = 128         # minibatch size
GAMMA = 0.99             # discount factor
TAU = 1e-3               # for soft update of target parameters
LR_ACTOR = 3e-4          # learning rate of the actor
LR_CRITIC = 3e-4         # learning rate of the critic
WEIGHT_DECAY = 0.0       # L2 weight decay
```

Within the model architecture, the number of units for the fully connected layers made a significant difference, and while I first used fc1_units=400, fc2_units=300, after several trials fc1_units=200, fc2 _units=400 nourished faster convergence. Additionally I used batch normalization for the 1st hidden layer.

I also used Leaky ReLU, which has a small slope for negative values, instead of altogether zero, with two main benefits:

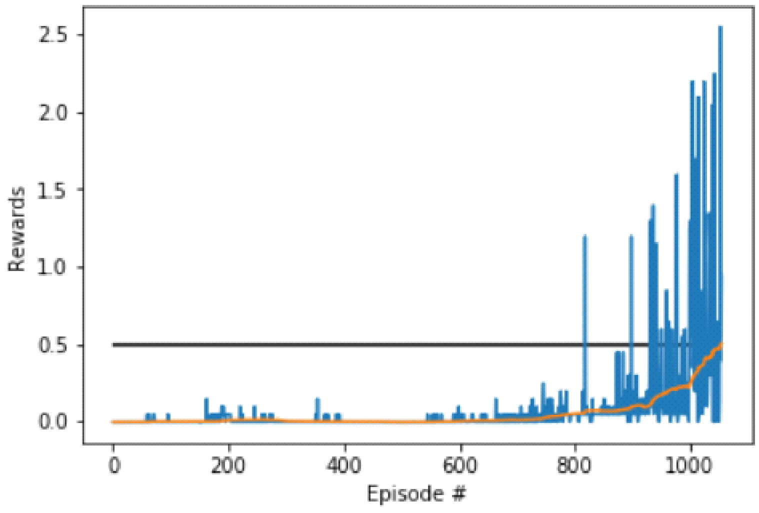
* It fixes the “dying ReLU” problem, as it doesn’t have zero-slope parts and
* It speeds up training. There is evidence that having the “mean activation” be close to 0 makes training faster. Unlike ReLU, leaky ReLU is more “balanced,” and may therefore learn faster.

One of the critical aspects within this multi-agent particular use case proved to be the learning rate for both the actor and the critic. Originally I tried a standard 1e-4 which had a slow convergence, tried 5e-4 as well which failed to converge (stuck in a local minima), and I settled with 3e-4 which proved consistent convergence over several runs.

Plot of Rewards

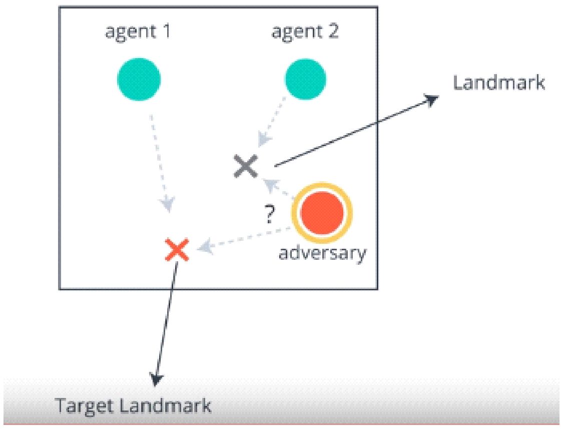
A plot of rewards per episode is included to illustrate that the agents get an average score of +0.5 (over 100 consecutive episodes, after taking the maximum over both agents).

```
Episode 100	Average Score: -0.00	Score: -0.005
Episode 200	Average Score: 0.01	Score: 0.04555
Episode 300	Average Score: 0.00	Score: -0.005
Episode 400	Average Score: -0.00	Score: -0.005
Episode 500	Average Score: -0.00	Score: -0.005
Episode 600	Average Score: 0.00	Score: 0.09555
Episode 700	Average Score: 0.01	Score: -0.005
Episode 800	Average Score: 0.05	Score: 0.0455
Episode 900	Average Score: 0.10	Score: 0.0455
Episode 1000	Average Score: 0.23	Score: 1.245
Episode 1055	Average Score: 0.50	Score: 0.9455
Environment solved in 1055 episodes!	Average Score: 0.50
```



Ideas for Future Work

A multi agent actor critic for mixed cooperative competitive environments was presented by Open AI, which represents the agents with particles, plus an additional set of landmarks. The agents cooperate to reach a target landmark. There is also an adversary aiming to reach the target landmark but it doesn't know which one of the landmarks is the target one.
The agents are rewarded based on the least distance to the landmark and penalized based on the distance between the adversary and the target landmark. The agents cooperate to spread the cost across all the landmarks.
A key idea is centralized training with decentralized execution, where during training each critic uses additional information as the states observed and action taken by the other agents, while the actor only has access to their agent's observations and actions. During the execution only the actors are used and thus all the observations and actions are available. A different reward structure for each agent is used, which facilitates the mixed use cases scenarios.



Another options is the Use of PPO (Proximal Policy Optimization), which is now the default reinforcement learning algorithm.

PPO strikes a balance between ease of implementation, sample complexity, and ease of tuning, trying to compute an update at each step that minimizes the cost function while ensuring the deviation from the previous policy is relatively small.

There is a new variant which uses a novel objective function by implementing a way to do a Trust Region update which is compatible with Stochastic Gradient Descent, and simplifies the algorithm by removing the KL penalty and the need to make adaptive updates.

$$L^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t)\,]$$

* ϑ is the policy parameter
* Et denotes the empirical expectation over timesteps
* rt is the ratio of the probability under the new and old policies, respectively
* At is the estimated advantage at time t
* ε is a hyperparameter, usually 0.1 or 0.2