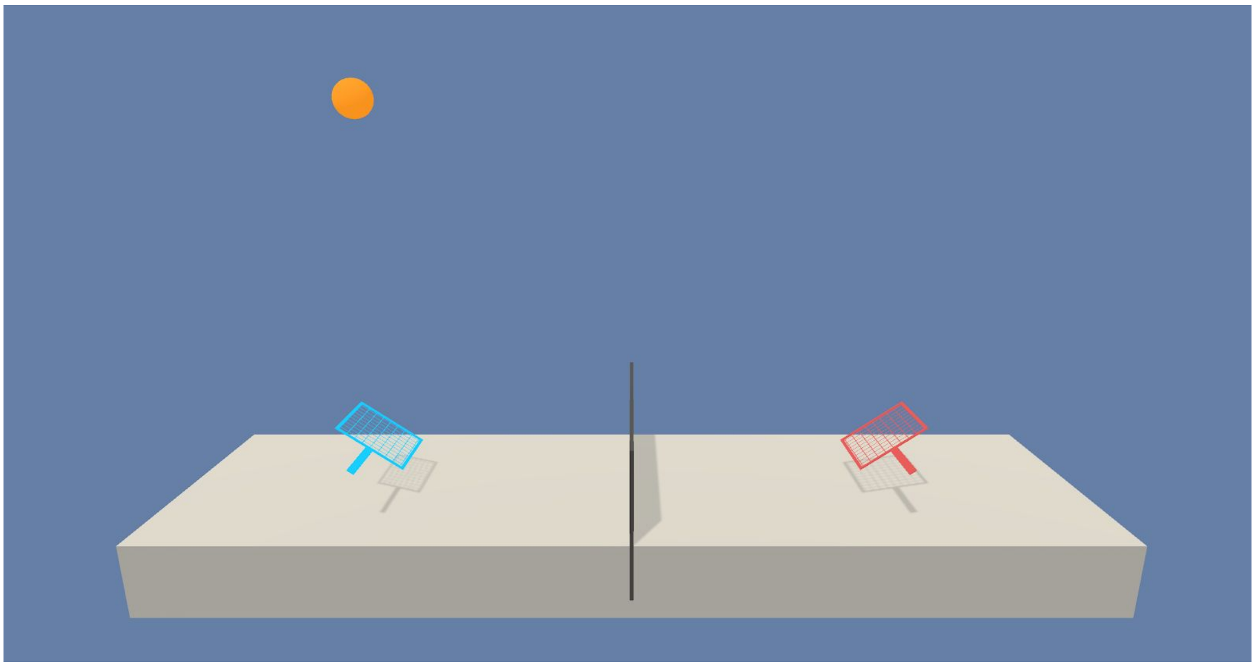
Udacity Deep Reinforcement Learning Nanodegree Program

Collaboration and Competition

Kellin Bershinsky



Introduction

In this 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. Two continuous actions are available, corresponding to movement toward (or away from) the net, and jumping. The goal of each agent is to keep the ball in play. 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 the ball hit the ground or hits the ball out of bounds, it receives a reward of -0.01. The task is episodic, and in order to solve the environment, the agent must get an average score of +0.5 (over 100 consecutive episodes, after taking the maximum over both agents). The environment is considered solved, when the average (over 100 episodes) of those scores is at least +0.5.

Learning Algorithm Used - DDPG

DDPG is a type of actor-critic method which can be seen as an approximate Deep Q-Network (DQN). The critic in DDPG is used to predict the maximum Q-value for the next state similar to the DQN algorithms estimate function. The main advantage that DDPG has over DQN is that it can be used in a continuous state and action space.

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

Figure 1: DDPG Algorithm[1]

[1] "Continuous control with deep reinforcement learning." https://arxiv.org/abs/1509.02971. Accessed 24 Jul. 2019.

Improvements Added:

Soft Update mitigates the slow learning problem of early state Q-learning in noisy environments caused by hard optimization policy bias.[2]

Replay Buffer for Experience Replay allows previous states to be revisited later in the learning process using updated agent weights to help mitigate temporal biasing.

Leaky ReLU helps with initial learning to prevent the “dead ReLU” problem caused by encountering only negative rewards since ReLU removes negative rewards from the learning function.

[2] "Taming the Noise in Reinforcement Learning via Soft Updates." 30 Mar. 2017, https://arxiv.org/pdf/1512.08562. Accessed 4 Aug. 2019.

Hyperparameters Used

BUFFER_SIZE = int(1e5) # replay buffer size

BATCH_SIZE = 128   # minibatch size

GAMMA = 0.99   # discount factor

TAU = 1e-3   # for soft update of target parameters

LR_ACTOR = 1e-5   # learning rate of the actor

LR_CRITIC = 1e-4   # learning rate of the critic

WEIGHT_DECAY = 0   # L2 weight decay

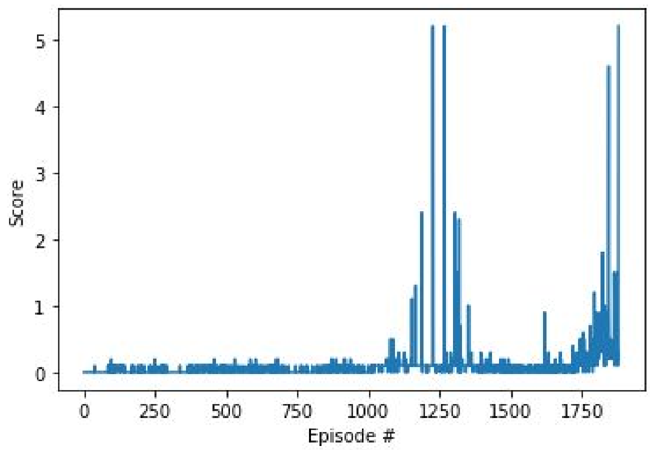
Model Architectures Used

The model architecture used for this project is an Actor-Critic network. The critic estimates the value function and the actor updates the policy in the direction recommended by the critic. Both the actor and critic have three linear hidden layers. Both the actor and critic use a leaky Rectifier Linear Unit (ReLU) activation function to ensure the gradients remain large enough through the hidden layers, but also allow a small positive gradient to prevent the Dying ReLU problem.[3]

[3] "Rectifier (neural networks) - Wikipedia." https://en.wikipedia.org/wiki/Rectifier_(neural_networks). Accessed 5 Aug. 2019.

Results - Plot of Rewards

```
Environment Solved! It took 1880 episodes!
Average Score: 0.52
```




```
Episode 100     Average Score: 0.01
Episode 200     Average Score: 0.04
Episode 300     Average Score: 0.02
Episode 400     Average Score: 0.01
Episode 500     Average Score: 0.05
Episode 600     Average Score: 0.02
Episode 700     Average Score: 0.04
Episode 800     Average Score: 0.01
Episode 900     Average Score: 0.05
Episode 1000    Average Score: 0.04
Episode 1100    Average Score: 0.10
Episode 1200    Average Score: 0.15
Episode 1300    Average Score: 0.22
Episode 1400    Average Score: 0.17
Episode 1500    Average Score: 0.07
Episode 1600    Average Score: 0.04
Episode 1700    Average Score: 0.07
Episode 1800    Average Score: 0.18
Episode 1880    Average Score: 0.52
```

The figure above shows the agents average score throughout the training process which included 1880 episodes. The table to the right of the plot shows the average score over 100 hundred episodes for every 100 episodes throughout the training session. Based on these results, it appears the environment was solved in 1880 episodes.

Ideas for Future Work

Trust Region Policy Optimization (TRPO)

TRPO is similar to natural policy gradient methods and is effective for optimizing large nonlinear policies such as neural networks.[4]

Algorithm 1 Policy iteration algorithm guaranteeing non-decreasing expected return η

Initialize π_0.
for $i = 0, 1, 2, \dots$ until convergence **do**
  Compute all advantage values $A_{\pi_i}(s, a)$.
  Solve the constrained optimization problem

$$\pi_{i+1} = \arg\max_{\pi} \left[L_{\pi_i}(\pi) - C D_{\mathrm{KL}}^{\max}(\pi_i, \pi) \right]$$

$$\text{where } C = 4\epsilon\gamma/(1-\gamma)^2$$

$$\text{and } L_{\pi_i}(\pi) = \eta(\pi_i) + \sum_s \rho_{\pi_i}(s) \sum_a \pi(a|s) A_{\pi_i}(s, a)$$

end for

Figure 2: TRPO Algorithm

Truncated Natural Policy Gradient (TNPG)

By computing an ascent direction that approximately ensures a small change in the policy distribution TNPG improves upon the REINFORCE algorithm.[5]

Distributed Distributional Deterministic Policy Gradients (D4PG)

D4DG adapts distributional reinforcement learning to the continuous control setting and combines it with distributed off-policy learning combined with improvement including N-step returns and prioritized experience replay.[6]

---

[4] "Trust Region Policy Optimization." https://arxiv.org/abs/1502.05477. Accessed 4 Aug. 2019.
[5] "Benchmarking Deep Reinforcement Learning for Continuous Control." 22 Apr. 2016, https://arxiv.org/abs/1604.06778. Accessed 24 Jul. 2019.
[6] "distributed distributional deterministic policy gradients - OpenReview." https://openreview.net/pdf?id=SyZipzbCb. Accessed 24 Jul. 2019.

Works Cited

Duan, et al. “Benchmarking Deep Reinforcement Learning for Continuous Control.” *ArXiv.org*, 27 May 2016, arxiv.org/abs/1604.06778.

Fox, et al. “Taming the Noise in Reinforcement Learning via Soft Updates.” *ArXiv.org*, 30 Mar. 2017, arxiv.org/abs/1512.08562.

P., Timothy, et al. “Continuous Control with Deep Reinforcement Learning.” *ArXiv.org*, 5 July 2019, arxiv.org/abs/1509.02971.

Schulman, et al. “Trust Region Policy Optimization.” *ArXiv.org*, 20 Apr. 2017, arxiv.org/abs/1502.05477.