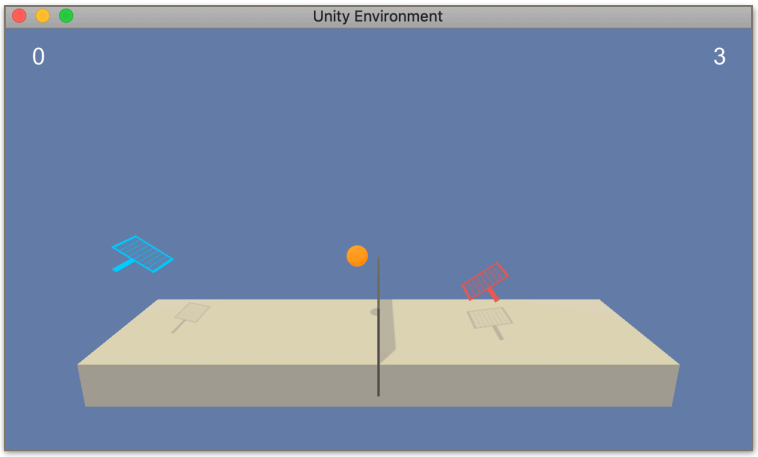
Student : Geoffrey Shmigelsky
School : Udacity
Class : Deep Reinforcement Learning
Assignment : Project #3 - Tennis Collaboration and Competition

Abstract

This paper is a summary report of my efforts to create a Deep Reinforcement Learning agent to solve the Tennis challenge.

The purpose of the simulation is to bounce a tennis ball over a net. The game is unusual in that two independent competitive agents have to work together and collaborate.



Problem Space

The search state space for this problem is significant: 24 input dimensions define the state for each agent; of which many are continuous. The action space consists of 2 continuous actions in a range (-1, 1) that are used to move the agent or jump.

A reward of +0.1 is given for game step where the ball is successfully lobbied over the net and -0.01 when the ball is dropped, ending the game.

My Approach

The best way to address this problem practically is with a deep neural network using an actor-critic approach. A deep neural network is by definition a universal function approximator - it can approximate any function; it should be possible to create a network that can solve the Tennis challenge.

With many actor-critic approaches are available, I utilized Deep Deterministic Policy Gradients (DDPG). According to the paper, Benchmarking Deep Reinforcement Learning for Continuous Control, Section 6, the *performance of the policy can degrade significantly during training.* DDPG's provide a good solution for continuous action spaces, but suffer from known training stability problems.

Acknowledgment to Udacity's class DDPG Source Code

The code used in this project is derived from the DDPG code in Project 2, Robotic Arms. The project 2 code is in turn based on the class supplied code in DDPG-Pendulum.

As per the suggestions in the class discussion, I modified the code to suit my needs to complete the Collaboration and Competition project. The source code for Udacity's supplied DDPG-Pendulum is at deep-reinforcement-learning.

Agent's Neural Network and Architecture

In Project 2's DDPG code, both the actor and critic networks had two fully connected layers with 400 and 300 neurons respectively. Testing yielded amazing results in this project with a network of this size using standard RELU and tanh activations.

Layer	Number of Nuerons
Input	24
Layer 1	400
Layer 2	300
Output	2

Rather than create two actor/critic agents, I chose a single training agent approach.

It seems logical to me that a single agent would be more effective than a collection of two distinct agents - they are both trying to solve the same problem.

Even if the state information and actions are somehow flipped or mirrored, a single neural-network should be able to handle it.

The software base from Project 2 is effectively unchanged for Project 3. It worked right out of the box, I only had to add in a max function for the reward calculation. The DDPG's agent and model were able to solve the Tennis challenge without any changes.

Agent Hyper-Parameters and Episodes

Buffer Size:	*100,000*
Batch Size:	*512*
Gamma:	*0.99*
Tau:	*1e-3*
LR Actor:	*1e-3*
LR Critic:	*1e-3*
Weight Decay:	*0.0*

The actor and critics Learning Rates to be identical to Project 2. However, when compared to the class supplied GitHub DDPG - they are not. If feel if the actor and critic are the same network shape, they should be trained the same way with the same learning rates, respectively.

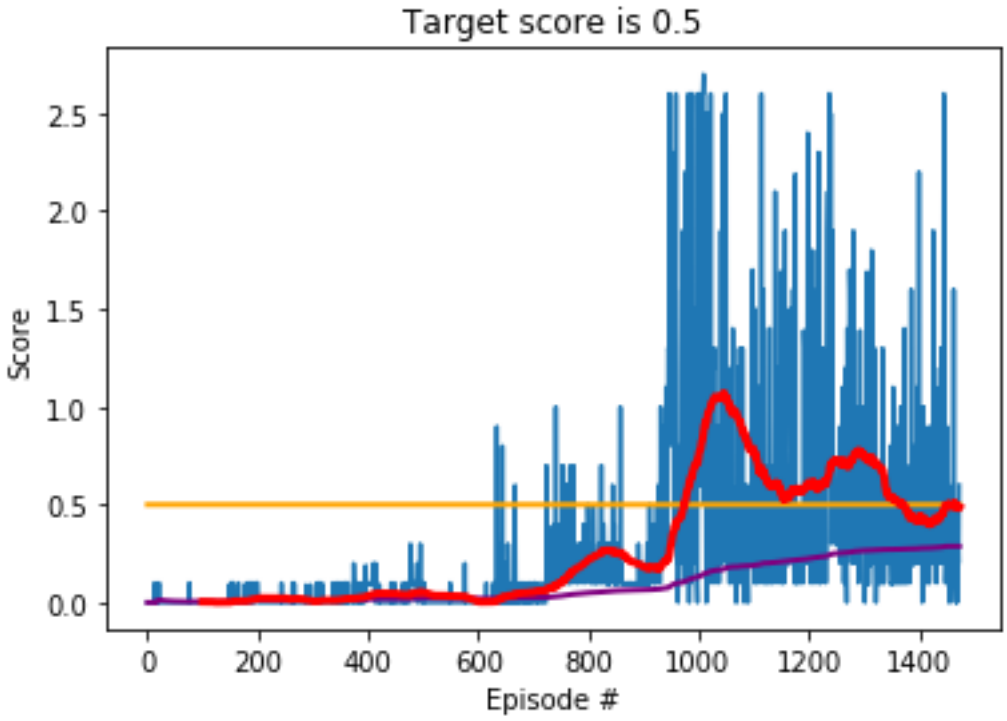
Training

The two agents max score crossed the average 0.50 target line at episode #973. This represents the mean score for the previous 100 episodes.

The agents continued to learn and excel at the game; reaching an average score above 1.0 near episode #1050. It even hit a top score of 2.5 multiple times.

More so - the DDPG agent demonstrate resilience by meeting the game objective for almost 400 additional episodes. When compared to the class provided benchmark, I think this result is outstanding.

In the diagram below, note how the agent falls into and out of local minima as the red line fluctuates.



Legend	
Blue	Max score for an episode, between two agents
Yellow	Target score of 0.50
Purple	Running average for all episodes
Red	Running 100 score average of the blue line.

Detailed Results

The agent solved Tennis with an average score of 0.50 over 100 episodes at episode #973. As you can see below, the agent started massive improvement around episode #900.

```
Episode:   0    Max Agents Score: 0.00   Deque Score: 0.00       Time: 0.1 seconds
Episode:  13    Max Agents Score: 0.10   Deque Score: 0.01       Time: 0.0 seconds
/home/geoffrey/Dropbox/RLND/Project3/ddpg_agent.py:106: UserWarning: torch.nn.utils.
_norm is now deprecated in favor of torch.nn.utils.clip_grad_norm_.
  torch.nn.utils.clip_grad_norm(self.critic_local.parameters(), 1)
Episode:  50    Max Agents Score: 0.00   Deque Score: 0.01       Time: 0.1 seconds
Episode: 100    Max Agents Score: 0.00   Deque Score: 0.00       Time: 0.1 seconds
Episode: 150    Max Agents Score: 0.00   Deque Score: 0.00       Time: 0.1 seconds
Episode: 200    Max Agents Score: 0.10   Deque Score: 0.02       Time: 0.2 seconds
Episode: 250    Max Agents Score: 0.00   Deque Score: 0.02       Time: 0.1 seconds
Episode: 300    Max Agents Score: 0.00   Deque Score: 0.01       Time: 0.1 seconds
Episode: 350    Max Agents Score: 0.00   Deque Score: 0.01       Time: 0.1 seconds
Episode: 400    Max Agents Score: 0.10   Deque Score: 0.03       Time: 0.2 seconds
Episode: 450    Max Agents Score: 0.00   Deque Score: 0.05       Time: 0.1 seconds
Episode: 500    Max Agents Score: 0.10   Deque Score: 0.05       Time: 0.2 seconds
Episode: 550    Max Agents Score: 0.00   Deque Score: 0.03       Time: 0.1 seconds
Episode: 600    Max Agents Score: 0.00   Deque Score: 0.00       Time: 0.1 seconds
Episode: 650    Max Agents Score: 0.10   Deque Score: 0.03       Time: 0.2 seconds
Episode: 700    Max Agents Score: 0.10   Deque Score: 0.05       Time: 0.2 seconds
Episode: 750    Max Agents Score: 0.30   Deque Score: 0.10       Time: 0.7 seconds
Episode: 800    Max Agents Score: 0.40   Deque Score: 0.22       Time: 1.0 seconds
Episode: 850    Max Agents Score: 0.09   Deque Score: 0.26       Time: 0.2 seconds
Episode: 900    Max Agents Score: 0.19   Deque Score: 0.18       Time: 0.5 seconds
Episode: 950    Max Agents Score: 1.90   Deque Score: 0.31       Time: 4.5 seconds
Episode: 973    Max Agents Score: 1.50   Deque Score: 0.50       Time: 3.8 seconds

=== SOLVED === at episode 973 with score 0.50

Episode: 1000   Max Agents Score: 2.20   Deque Score: 0.77       Time: 5.4 seconds
Episode: 1050   Max Agents Score: 2.60   Deque Score: 1.05       Time: 6.8 seconds
Episode: 1100   Max Agents Score: 1.20   Deque Score: 0.76       Time: 3.4 seconds
Episode: 1150   Max Agents Score: 0.10   Deque Score: 0.55       Time: 0.6 seconds
Episode: 1200   Max Agents Score: 0.20   Deque Score: 0.60       Time: 0.8 seconds
Episode: 1250   Max Agents Score: 0.40   Deque Score: 0.72       Time: 1.4 seconds
Episode: 1300   Max Agents Score: 0.29   Deque Score: 0.74       Time: 1.0 seconds
Episode: 1350   Max Agents Score: 0.09   Deque Score: 0.53       Time: 0.3 seconds
Episode: 1400   Max Agents Score: 2.20   Deque Score: 0.44       Time: 7.2 seconds
Episode: 1450   Max Agents Score: 0.90   Deque Score: 0.50       Time: 3.4 seconds
Episode: 1473   Max Agents Score: 0.60   Deque Score: 0.49       Time: 2.2 seconds
CPU times: user 18min 41s, sys: 1min 30s, total: 20min 11s
Wall time: 21min 54s
```

The agent continued to run for another 500 episodes to investigate training stability. Note the Deque score continues to be strong, often scoring over the target.

I also investigated the scores from either agent; they were almost identical. Indicating that the model was able to find a robust solution for both sides of the net. However, when you watch the agents play each other, they have their own style of play.

Future Enhancements

There are at least three ways to improve the agents performance:

Neural Network Configurations

> There are likely other neural network sizes and shapes to explore that may be more efficient at approximately the actor or critic function. Given how simple the environment is versus Project 2, I think a small network may be the way to go.

Hyper Parameters

> A clear choice that would be easy to implement would be a grid search or random search of the agent's hyper-parameters (Learning Rate, Gamma, Epsilon). There may be some combination that would yield a more stable learning curve. I have used SKLearn's GridSearchCV function in the past to brute force search the hyper parameter space with good success.

Proximal Policy Optimization

> A different training algorithm entirely may yield better results or a more stable learning curve.

Summary

I was amazed that Project 2's agent code, model and DDPG ran almost right out of the box to successful completion in Project 3. Even with trying different configurations, almost every single one was able to solve the game.

It is quite impressive that DDPG can find a solution to vastly different problem so reliably on an almost identical code base. It is a statement to the power of Reinforcement Learning algorithms.