Collaboration and Competition Project

Udacity Deep Reinforcement Learning Nano-Degree

Jeremy Vila

December 21, 2018

1 Introduction

In this project, I overview a implementation of the multi agent deep deterministic policy gradients (MADDPG) algorithm to solve the collaboration and control project in Udacity's Deep Reinforcement (RL) Learning Nano-degree. After reviewing the implementation, we demonstrate that the trained MADDPG model achieves the objective of the project two agents to play a game of virtual tennis. A reward of $+0.1$ is provided to an agent if it hits the ball over the net. If an agent lets the ball hit the floor, their side of the table, or hits the ball out of play it gets a score of -0.01. The project is considered complete if an average score of $+0.5$ is achieved over a window of 100 episodes (after a maximum is taken between the scores of the agents). Therefor the goal is to train a pair of agents to play with each other as long as possible.

There are 8 variables in the observation space corresponding to position and velocity of the ball and racket. There are 2 actions, corresponding to moving towards the net, and jumping. These actions are bounded between -1 and 1.

2 Learning Algorithm

This implementation employs the Multi Agent Deep Deterministic Policy Gradients (MADDPG) method to solve the Continuous Control project on the Reacher robot. This MADDPG implementation contains two DDPG agents, each using the SAME models. Because of the symmetric nature of this problem (e.g., moving towards and away from the net) and the fact that they are trying to achieve the same objective, the two agents can update the same DDPG model.

Specifically, each DDPG agent has two separate networks, one for the actor and one for the critic. The algorithm alternates between updating the actor and critic networks, each leveraging a related target network. This target network is updated every episode (sometimes called a "soft update"). I considered updating the network every 10 episodes, but found that it did not improve results, and only slowed down training

All hyperparameters, e.g., the number of layers and the number of hidden neurons, were selected using a hand-tuning procedure. In general, we found that very simple deep networks with a small number of hidden neurons (e.g. 50 to 100 were the best performing. Likely, large models were overfitting to the problem, or getting stuck in local minima. The best performing discount factor was found to be $\gamma = 0.95$. I also found that the critic's learning rate (step size) needed to be set relatively large as $1e-3$, while the actor's needed to be smaller at $1e-4$. Ornstein-Uhlenbeck noise was considered, as proposed in the original DDPG paper (and whose placeholder is found in the code, but this author found that it caused convergence issues, even with extensive hyperparameter tuning. Batch size was set to 256, though it did not have much of an impact on results. Lastly, the loss function was set to be a smoother ell_1 loss, as the mean squared error had a hard time getting above a reward of 0.1

2.1 Critic Network

The critic network $Q(s, a|\theta_{crit})$ is defined as exactly in Q-networks. In this implementation, the network is updated via a temporal difference (TD) method, as was done in the previous project. A replay buffer was also utilized, so that randomized experiences from previous examples can be used to learn at each update step.

The critic network contains 4 fully connected layers, with the first layer acting only on the state inputs. The subsequent hidden layer is concatenated with the action space, and followed by two more fully connected layers. The number of hidden neurons for each of the hidden layers is 50, 100, and 50 respectively. All hidden layers contain ReLU activation functions, and the final output has a linear output.

2.2 Actor Network

The actor network is a deep neural network to approximate the action that maximizes policy, e.g.,

$$\mu(s;\theta_{act}) = \text{argmax}_a \pi\left(a|s,\theta_{act}\right). \tag{1}$$

This network consists of three fully conected layers of with the first two layers having 50 and 100 hidden neurons, respectively, and each activated by a relu layer. The final layer has an output of 2 possible actions, with a "tanh" activation function because the actions are clipped to $[-1, 1]$.

3 Results

The results for the training process is shown in Figure 1. The training process was completed after 1090 episodes, where it averaged a score great than +0.6 over the previous 100 episodes (with a maximum taken over the agents). One interesting observation is that though the average score increases across episodes, so does the variance and volatility. Another observation was that it took a long time 600 epochs before they started learning something useful. In general, it was difficult to:

1. Hit the ball over the net correctly. This seemed to require a specific action. Exploration is needed to find this action. Once found, it needs to exploit it consistently

The second agent to return the ball. Again, exploration is needed here.

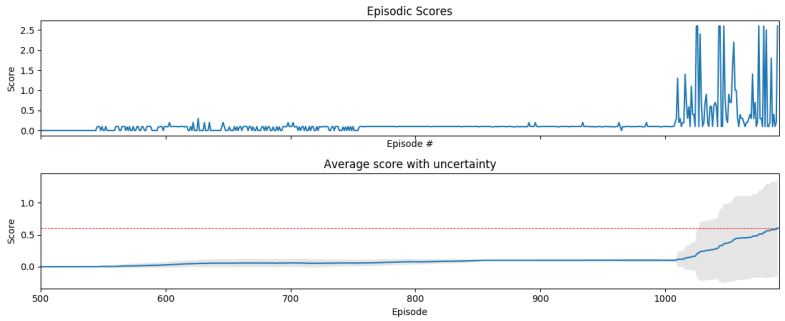

Figure 1: Episode number vs. score.

4 Future Improvements

In the proposed approach, we used a standard DDPG model to train the actor and the critic. More advanced policy gradient methods such as PPO that lend themselves bettter to this problem, might yield better results or faster convergence to a solution.

Despite being in a competitive environment, this framework lends itself to other synchronous and asynchronous methods as well:

1. A2C: The Advantage Actor Critic method with synchronous weight updates. With this algorithm, each of the parallel agents wait to update the model weights

2. A3C: The Asynchronous Advantage Actor Critic Method uses the same procedure, but periodically updates the model weights once each agent has an update

More advanced topics such as the Trust Region Policy Optimization (TRPO) and Truncated Natural Policy Gradient (TNPG) should also be considered in the future.

Finally, we excluded the noise process, which generally helped agents reach a score of 0.1. However, this noise process caused the reward to plateu. Perhaps a damping procedure on the noise should have been considered.