Continuous Control Project

Udacity Deep Reinforcement Learning Nano-Degree

Jeremy Vila

November 20, 2018

1 Introduction

In this project, I overview my particular implementation of the deep deterministic policy gradients (DDPG) algorithm to solve the continuous control project in Udacity's Deep Reinforcement (RL) Learning Nano-degree, and show the results. In this project, the goal is to control a robotic arm can move to target locations. A reward of +0.1 is provided for each step that the agent's hand is in the goal location. Therefore, the goal is to get the hand to the target as quickly as possible, then remain in that location

There are 33 variables in the observation space corresponding to position, rotation, velocity, and angular velocities of the arm. There are 4 actions, corresponding to torque applicable to two joints. These actions are bounded between -1 and 1.

We apply the DDPG to solve the single "Reacher" robot example. We did not consider the distributed training example of controlling many arms in parallel due to computational constraints.

2 Learning Algorithm

This implementation employs the Deep Deterministic Policy Gradients (DDPG) method to solve the Continuous Control project on the Reacher robot. Specifically, I consider 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 only once every 10 episodes (sometimes called a "soft update").

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 a relatively large (compared to observation or action space) was needed for the hidden layer neurons. The discount factor was set as $\gamma = 0.95$. I also found that the learning rate (step size) needed to be set relatively large as $1e-3$. Noise was added using a Ornstein-Uhlenbeck process, as is done in the original DDPG paper. Batch size was set to 256, though it did not have much of an impact on results.

2.1 Critic Network

The critic network $Q\left(s, a|\theta_{crit}\right)$ 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 256, 256, and 128 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 256 and 128 hidden neurons, respecitively, and each activated by a relu layer. The final layer has an output of 4 possible actions, with a "tanh" activation function because the actionas are clipped to $[-1, 1]$.

3 Results

The results for the training process is shown in Figure 1. The training process was completed after 216 episodes, where it averaged a score great than +30 over the previous 100 episodes. I let the training go on until episode 236 where an average score of +32 was achieved so that I can monitor behavior after convergence. One interesting observation is that though the average score increases across episodes, so does the variance and volatility.

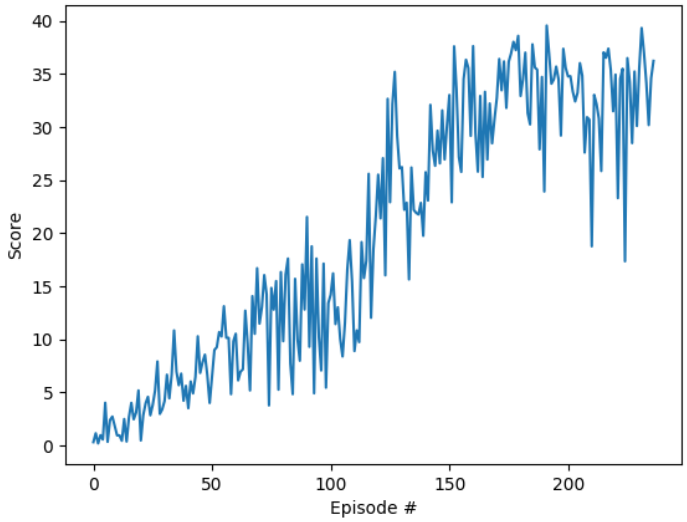

Figure 1: Episode number vs. score.

4 Future Improvements

In the proposed approach, we used a standard DDPG model to train the actor and the critic. In general, there are a suite of policy gradient methods that lend themselves to this problem, some that might yield better results or faster convergence to a solution. One possible way to get faster convergence is to solve the distributed problem with multiple robotic arms in parallel. This framework lends itself to synchronous and asynchronous methods:

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 can consider using alternate sensor data entirely, e.g., a camera. Image and video data can offer a richer observation space. In this scenario, a convolutional neural network is likely to be a better choice of model for the q function, as they are shown to reliably handle image data.