Twin Delayed Deep Deterministic Policy Gradients for Continuous Control Report

Dalton J. Schutte

Background

The Deep Deterministic Policy Gradients (DDPG) algorithm was first proposed by Lillicrap et al. in a 2015 paper, Continuous Control with Deep Reinforcement Learning[1], that utilized a combination of an Actor network to choose a policy given a state and a Critic, or Value, network to evaluate the action and state. While this algorithm can learn continuous control tasks, it tends to be brittle to the choice of hyperparameters and the Actor can learn to "game" the Critic while not learning an optimal policy. Fujimoto et al. proposed a modified version of DDPG in their 2018 paper, Addressing Function Approximation Error in Actor-Critic Methods[2], that incorporated Clipped Double Q-Learning to help reduce overestimation bias, adding noise to the target policy to smooth the policy, and delaying policy updates to reduce the effects of error accumulation. This new algorithm, Twin Delayed DDPG (T3DPG) demonstrated better performance on continuous control tasks in their research.

Problem

In this project, the goal was to train an agent to control one, or 20, robotic arm(s) to keep the end inside a moving region. At each time-step the agent received +0.1 reward if the end of the arm was within the region and no reward otherwise. In the 20-arm instance, the reward is a vector of length 20 each entry being 0 or +0.1. The environment states have 33 values and the agent can set an action vector containing four values, fixed between -1 and 1, at each time-step with each value being the torque in the two joints.

Solution

The 20-arm environment was used for training, with a single agent selecting actions for each of the 20 arms. The environment runs for a maximum of 1000 time-steps so each episode consists of 20,000 time-steps experienced by the agent.

Initially, the DDPG algorithm was used but, in many attempts, across many hyperparameter configurations and random seeds, the agent would quickly learn a good policy but then wander into less well-performing policies and not recover. Per the DDPG paper, batch normalization was used for both the Actor and Critic networks but after exhausting other hyperparameters research was done and a 2019 paper, CrossNorm: On Normalization for Off-Policy TD Reinforcement Learning[3], found that batch normalization often worsened the performance of DDPG. After removing batch normalization from the Actor and Critic networks there was a noticeable improvement in the agent's stability.

[1] https://arxiv.org/pdf/1509.02971.pdf
[2] https://arxiv.org/pdf/1802.09477.pdf
[3] https://arxiv.org/pdf/1902.05605.pdf

During the additional research, one of the most common critics of the DDPG algorithm was how sensitive it is to the choice of learning rate when using the Adam optimizer. Further research revealed that AMSGrad[4] improved the convergence of Adam and was implemented in Pytorch. It was also found in a 2019 paper, Accelerated Methods for Deep Reinforcement Learning[5], that batch sizes of 512 often produced better policies. However, with the removal of batch normalization, the addition of AMSGrad to Adam, and batch size of 512, DDPG was still struggling to learn and maintain an optimal policy.

The TD3PG algorithm was then used the policy did not obtain high rewards as quickly but did seem more stable and wandered into less well-performing policies less often. The TD3PG algorithm chooses the next action Q-values using the minimum of the output from both of the Critic networks. The loss is sum of the MSE of the expected value of the Critic network and the target values for both Critic networks.

Hyperparameters

The replay memory was initialized to a capacity of 200,000 of the most recent experiences and would sample 512 experiences uniformly without replacement at each update. The learning rate for the Adam optimizer was set to 0.003 and 0.0003 for the Critics and Actor, respectively, the discount factor to 0.99, the update interpolation rate to 0.005, the update frequency to 15 updates every 10 time-steps with the policy and target networks being updated only every 2 time-steps, epsilon (scaling factor for the Ornstein-Uhlenbeck noise added to encourage exploration) minimum to 0.01, and epsilon was exponentially decreased using the following:

$$\varepsilon * e^{-0.005*ep}$$

Where *ep* is the episode number beginning at zero.

For the Ornstein-Uhlenbeck noise, mu was 0, theta was 0.15, and sigma 0.2. The noise added to the target actions for policy smoothing was drawn from a normal distribution centered on 0 with standard deviation of 0.2.

Gradient norms were clipped to between -1 and 1 prior to each gradient update.

The random seed was set to 628.

Actor Network Architecture

Layer (Activation)	Input Dimension	Output Dimension
Fully Connected (SeLU[6])	33	256
Fully Connected (SeLU)	256	128
Output (Tanh)	128	4

[4] https://openreview.net/pdf?id=ryQu7f-RZ
[5] https://arxiv.org/pdf/1803.02811.pdf
[6] https://arxiv.org/abs/1706.02515

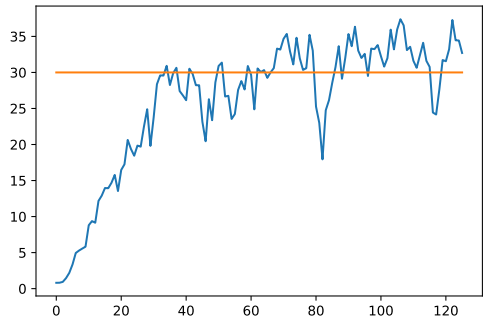
Critic Network Architecture

Layer (Activation)	Input Dimension	Output Dimension
Fully Connected (SeLU)	33	256
Fully Connected (SeLU)	256 + 4	128
Output	128	1

Weights were initialized in the same manner as the DDPG paper.

Results

The agent solved the task in 127 episodes and attained an average of 30.05 over the last 100 episodes. The orange line is at a score of 30.



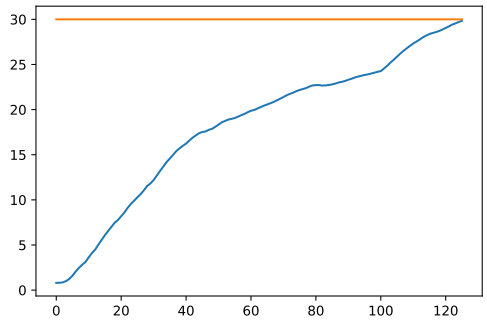


Figure 1 (left) shows the reward after each episode for the TD3PG algorithm and Figure 2 (right) shows the running mean over a window of 100 episodes for the TD3PG algorithm. The orange line in both figures is the target reward of 30

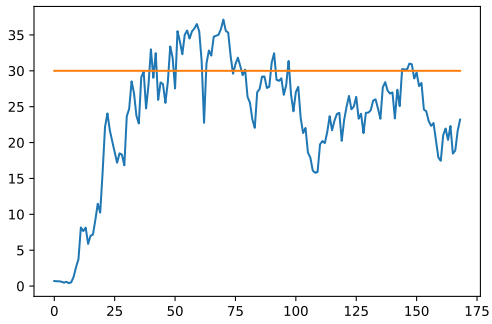


Figure 3 shows the reward after each episode for the DDPG algorithm. It can be seen that once the policy wanders away from the optimal policy it struggles to recover.

Conclusion

To improve the training, various improvements could be used. It would be interesting to see how improvements made for Deep Q-Networks might be applied to the Critic networks here. Double Q-Learning was modified for the TD3PG paper, similar efforts could be made with distributional reinforcement learning, noisy networks, and dueling network architectures. Prioritized experience replay would also likely improve the learning for this algorithm.