Deep Reinforcement Learning- Continuous Control

Udacity- Project 2

Unity Reacher Environment

Introduction

In the report for the second project required for partial fulfillment of requirements pertaining to Udacity's Deep Reinforcement Learning Nanodegree, a Deep Deterministic Policy Gradient based Agent was implemented to solve Unity's Reacher Environment with 20 identical agents. The grading rubric calls for the report to delve into the Learning Algorithm, Reward Plots and Ideas for Future work. Details about implementation and the environment could be found in the 'README.md' file of the home repository.

Algorithms

Policy gradient methods are a type of reinforcement learning techniques that rely upon optimizing parametrized policies with respect to the expected return (long-term cumulative reward) by gradient descent. They do not suffer from many of the problems that have been marrying traditional reinforcement learning approaches such as the lack of guarantees of a value function, the intractability problem resulting from uncertain state information and the complexity arising from continuous states & actions.

The general goal of policy optimization in reinforcement learning is to optimize the policy parameters $\theta \in R^K$ so that the expected return,

$$J(\theta) = E\left\{\sum_{k=0}^{H} a_k r_k\right\}$$

is optimized where a_K denotes time-step dependent weighting factors, often set to $a_K = \gamma^K$ for discounted reinforcement learning (where γ is in [0,1]) or $a_K = 1/H$ for the average reward case. For real-world applications, we require that any change to the policy parameterization has to be smooth as drastic changes can be hazardous for the actor as well as useful initializations of the policy based on domain knowledge would otherwise vanish after a single update step. For these reasons, policy gradient methods which follow the steepest descent on the expected return are

the method of choice. These methods update the policy parameterization according to the gradient update rule,

$$\theta_{h+1} = \theta_h + \alpha_h \nabla_\theta J|_{\theta=\theta_h},$$

Where $a^h \in \mathbb{R}^+$ denotes a learning rate and h ∈ {0,1,2,.....} the current update number.

Deep Deterministic Policy Gradient (DDPG)

DDPG uses four neural networks: a Q network, a deterministic policy network, a target Q network, and a target policy network. The Q network and policy network is very much like simple Advantage Actor-Critic, but in DDPG, the Actor directly maps states to actions (the output of the network directly the output) instead of outputting the probability distribution across a discrete action space.The target networks are time-delayed copies of their original networks that slowly track the learned networks. Using these target value networks greatly improve stability in learning.

The Pseudo-Code of the algorithm is,

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

Taken from "Continuous Control With Deep Reinforcement Learning" (Lillicrap et al, 2015)

Implementation Details

The Actor Network has two hidden layers and an output layer. A batch-normalization step is also added after the first hidden layer. ReLu activation is used for after the batch normalization layer and also after the second hidden layer. Tanh activation is used after the output layer.

The Critic Network also has two hidden layers and an output layer. Similar to the Actor Network, a batch-normalization added after the first hidden layer and ReLu activation is used for after the batch normalization layer and after the second hidden layer. The initial input to the Critic Network are the states, actions are later added by concatenating the output from the first ReLu activation layer with the actions and passing the concatenated entity to the next hidden layer. No activation is applied after the output layer.

The following hyperparameters were used for the agents,

1. BUFFER_SIZE = 1000000
2. BATCH_SIZE = 128
3. GAMMA = 0.99
4. TAU = 1e-3
5. LR_ACTOR = 1e-3
6. LR_CRITIC = 1e-3
7. WEIGHT_DECAY = 0
8. LEARN_EVERY = 20
9. LEARN_NUM = 10
10. OU_SIGMA = 0.2
11. OU_THETA = 0.15
12. EPSILON = 1.0
13. EPSILON_DECAY = 1e-6

Other hyperparameters pertaining to the training process could be found in the ‘Continuous_Control.ipynb’ notebook.

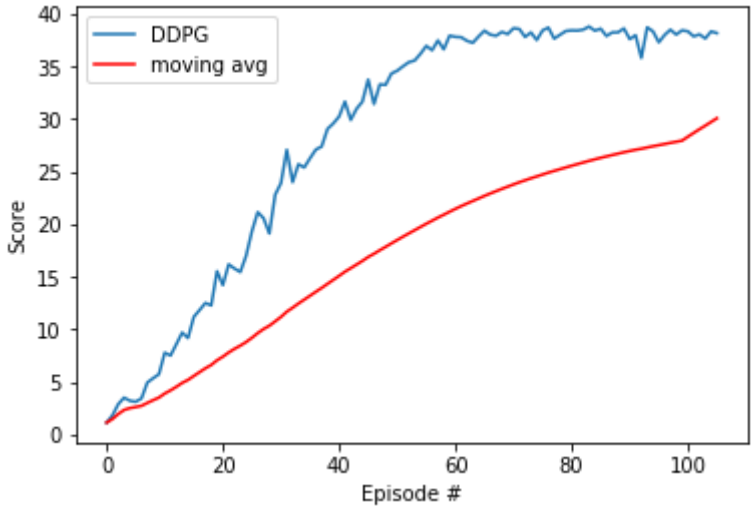
Reward Plots

The algorithm converged in 106 episodes,

```
Episode 83 (239 sec)  --      Min: 35.0      Max: 39.6      Mean: 38.5      Mov. Avg: 25.9
Episode 84 (238 sec)  --      Min: 35.8      Max: 39.6      Mean: 38.8      Mov. Avg: 26.0
Episode 85 (238 sec)  --      Min: 36.6      Max: 39.6      Mean: 38.4      Mov. Avg: 26.2
Episode 86 (238 sec)  --      Min: 37.2      Max: 39.6      Mean: 38.6      Mov. Avg: 26.3
Episode 87 (238 sec)  --      Min: 35.5      Max: 39.4      Mean: 37.9      Mov. Avg: 26.5
Episode 88 (238 sec)  --      Min: 36.2      Max: 39.6      Mean: 38.2      Mov. Avg: 26.6
Episode 89 (239 sec)  --      Min: 36.4      Max: 39.5      Mean: 38.3      Mov. Avg: 26.7
Episode 90 (240 sec)  --      Min: 36.0      Max: 39.6      Mean: 38.6      Mov. Avg: 26.9
Episode 91 (239 sec)  --      Min: 34.3      Max: 39.3      Mean: 37.6      Mov. Avg: 27.0
Episode 92 (239 sec)  --      Min: 34.7      Max: 39.4      Mean: 38.0      Mov. Avg: 27.1
Episode 93 (240 sec)  --      Min: 30.9      Max: 38.6      Mean: 35.8      Mov. Avg: 27.2
Episode 94 (239 sec)  --      Min: 35.4      Max: 39.6      Mean: 38.7      Mov. Avg: 27.3
Episode 95 (238 sec)  --      Min: 36.2      Max: 39.5      Mean: 38.4      Mov. Avg: 27.4
Episode 96 (238 sec)  --      Min: 32.5      Max: 39.6      Mean: 37.3      Mov. Avg: 27.5
Episode 97 (238 sec)  --      Min: 34.3      Max: 39.5      Mean: 38.0      Mov. Avg: 27.6
Episode 98 (239 sec)  --      Min: 36.3      Max: 39.6      Mean: 38.5      Mov. Avg: 27.8
Episode 99 (240 sec)  --      Min: 33.8      Max: 39.6      Mean: 38.0      Mov. Avg: 27.9
Episode 100 (239 sec)  --     Min: 36.0      Max: 39.6      Mean: 38.4      Mov. Avg: 28.0
Episode 101 (239 sec)  --     Min: 36.2      Max: 39.6      Mean: 38.4      Mov. Avg: 28.3
Episode 102 (240 sec)  --     Min: 34.3      Max: 39.6      Mean: 37.9      Mov. Avg: 28.7
Episode 103 (239 sec)  --     Min: 34.2      Max: 39.4      Mean: 38.1      Mov. Avg: 29.0
Episode 104 (239 sec)  --     Min: 33.9      Max: 39.4      Mean: 37.7      Mov. Avg: 29.4
Episode 105 (239 sec)  --     Min: 36.6      Max: 39.6      Mean: 38.3      Mov. Avg: 29.7
Episode 106 (239 sec)  --     Min: 35.4      Max: 39.5      Mean: 38.2      Mov. Avg: 30.1

Environment SOLVED in 6 episodes!       Moving Average =30.1 over last 100 episodes
```

The reward and moving average of rewards for a window size of 100 are depicted in the plot,



Ideas for Future work

1. Prioritization for replay buffer
2. Try Trust Region Policy Optimization.
3. Try Distributed Distributional Deterministic Policy Gradient.
4. Hyper parameter optimization
5. Experiment with different network architecture and activation layers