Report

26 March 2023

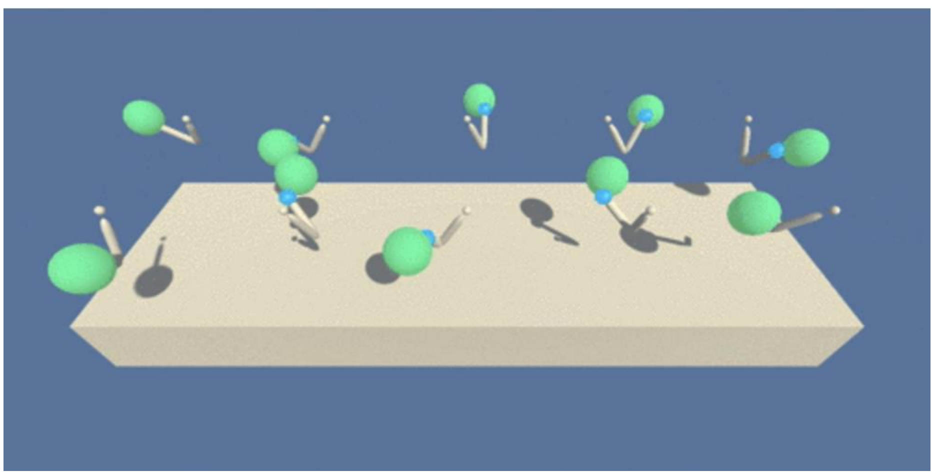
Introduction

This project is on the **Reacher** environment.

In this environment, a double-jointed 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. Thus, the goal of your agent is to maintain its position at the target location for as many time steps as possible.

The observation space consists of 33 variables corresponding to position, rotation, velocity, and angular velocities of the arm. Each action is a vector with four numbers, corresponding to torque applicable to two joints. Every entry in the action vector should be a number between -1 and 1.

The task is episodic, and in order to solve the environment, the agents must get an average score of +30 over 100 consecutive episodes.



Learning Algorithm

The algorithm used in this project is a Deep Deterministic Policy Gradient (DDPG), an algorithm in actor-critic method.

Value-based methods have low bias but high variance, whereas policy-based methods have low variance but high bias. Actor-critic methods can be seen as a trade-off between value-based and policy-based methods.

Actor-critic methods aim to strike a balance between bias and variance by combining the benefits of both value-based and policy-based methods. By learning both the policy and value function, actor-critic methods can reduce the bias of policy-based methods and the variance of value-based methods, resulting in more stable and efficient learning.

A DDPG has two networks, actor and critic. The goal of the actor is to learn a policy that maximizes the expected return over time. The critic is responsible for estimating the value function of the policy. The value function estimates the expected return for a given state. The goal of the critic is to learn an accurate value function that can guide the actor towards actions that lead to higher expected returns.

Both actor and critic have two networks, local and target. They are to stabilize the learning process and prevent divergence. The local networks are the main networks used for action selection and value estimation. Target networks are used to estimate the target values during the learning process. The weights of the target networks are periodically updated with a fraction of the weights of the corresponding local networks, providing a more stable target for learning. This reduces the correlation between successive observations and results in a more stable learning process.

The hyperparameter are as follows:

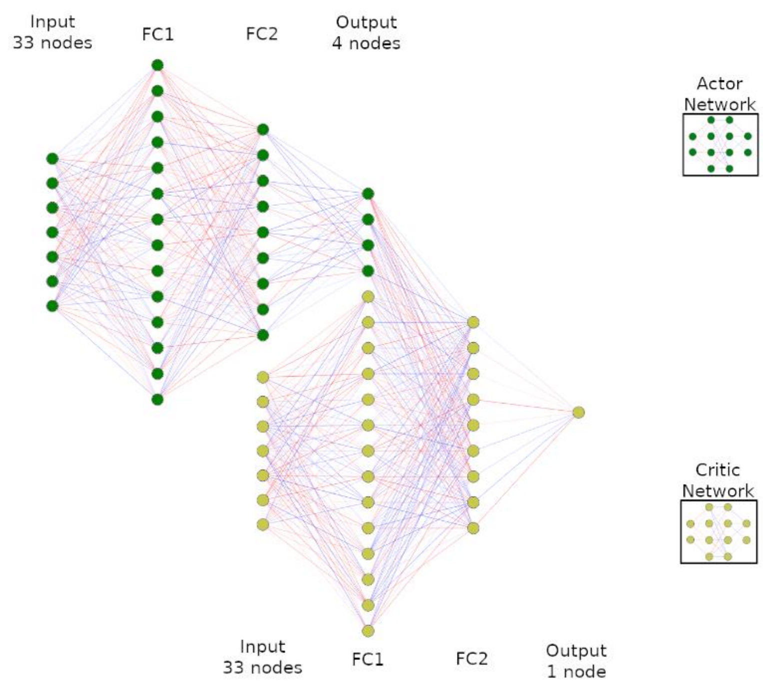
```
BUFFER_SIZE = int(1e5)  # replay buffer size
BATCH_SIZE = 128        # minibatch size
GAMMA = 0.99            # discount factor
TAU = 1e-3              # for soft update of target parameters
LR_ACTOR = 1e-3         # learning rate of the actor
LR_CRITIC = 1e-3        # learning rate of the critic
WEIGHT_DECAY = 0        # L2 weight decay
```

The network:

The input to the actor network is the observation space consisting of 33 variables and the output of 4 numbers is the predicted best action for that observed state.

The input to the critic network is also the observation space. The input to the critic's second hidden layer is the output of the critic's first hidden layer and the output of the actor's network. The output of the overall network is the prediction of the target value based on the given state and the predicted best action. Refer to figure below where the image obtained from this link. FC1 and FC2 are both equal to 128.

In short, critic calculates the optimal action-value function *Q(s, a)* by using the action from actor's deterministic optimal policy.



DDPG with Actor and Critic networks [link].

Actor:
First layer: input size = **33**, output size = 128
Second layer: input size = 128, output size = 128
Third layer: input size = 128, output size = **4**

Critic:
First layer: input size = 33, output size = 128
Second layer: input size = **132**, output size = 128
Third layer: input size = 128, output size = **1**

Plot of Rewards

The average reward for all 20 agents over 100 episodes is **30.06** achieved in **357 episodes**.

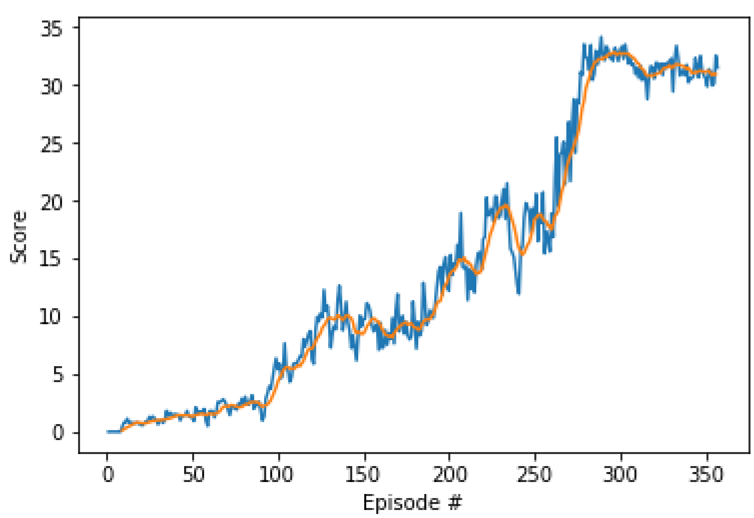
```
with active_session():
    # do long-running work here
    scores = ddpg()
```

```
Episode 100     Average Score: 1.68
Episode 200     Average Score: 9.09
Episode 300     Average Score: 21.73
Episode 357     Average Score: 30.06
Solved in 357 episodes.         Average Score: 30.06
```

```
fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(np.arange(1, len(scores)+1), scores)

rolling_window = 10
rolling_mean = pd.Series(scores).rolling(rolling_window).mean()
plt.plot(rolling_mean);

plt.ylabel('Score')
plt.xlabel('Episode #')
plt.show()
```



Future Works

To improve the performance of the agent, below are some of the possible future works.

- Hyper-parameters
  To investigate optimal parameters to solve a given environment. This is very important especially for the single agent environment (see [Additional Remarks] below).
- Prioritized Experience Replay
  This technique learns more efficiently by replaying important transitions more frequently unlike in the current implementation where all transitions are sampled uniformly.
- Implement Other algorithms
  Some of the possible algorithms for implementation includes
  - Proximal Policy Optimization (PPO),
  - Distributed Distributional Deterministic Policy Gradients (D4PG),
  - Trust Region Policy Optimization (TRPO),
  - Asynchronous Advantage Actor-Critic (A3C), and
  - Advantage Actor-Critic (A2C).

Additional Remarks

- It is easier to solve multi agents than a single agent. Many attempts have been made to solve a single agent. I have varied the network size and hyper-parameters especially the learning rate but still unable to solve. I have used a lot of GPU hours in those attempts.

- This is one of those attempts to solve a single agent.

```
with active_session():
    # do long-running work here
    scores = ddpg()
```

```
Episode 100     Average Score: 1.78
Episode 200     Average Score: 6.34
Episode 300     Average Score: 13.47
Episode 400     Average Score: 22.21
Episode 500     Average Score: 24.89
Episode 600     Average Score: 24.46
Episode 700     Average Score: 24.69
Episode 800     Average Score: 24.50
Episode 900     Average Score: 24.62
Episode 1000    Average Score: 24.64
Episode 1100    Average Score: 24.88
Episode 1200    Average Score: 23.90
Episode 1300    Average Score: 24.42
Episode 1400    Average Score: 24.63
Episode 1500    Average Score: 24.14
Episode 1600    Average Score: 23.87
Episode 1700    Average Score: 24.80
Episode 1800    Average Score: 23.86
Episode 1900    Average Score: 23.16
Episode 2000    Average Score: 23.94
Episode 2100    Average Score: 23.89
Episode 2200    Average Score: 24.16
Episode 2300    Average Score: 23.51
Episode 2400    Average Score: 23.43
Episode 2500    Average Score: 23.25
Episode 2600    Average Score: 23.89
Episode 2700    Average Score: 22.96
Episode 2739    Average Score: 23.22
```

- This is another attempt to solve a single agent.

```
scores = ddpg()
```

```
Episode 100      Average Score: 2.00
Episode 200      Average Score: 6.81
Episode 300      Average Score: 14.03
Episode 400      Average Score: 19.33
Episode 500      Average Score: 22.09
Episode 600      Average Score: 24.08
Episode 700      Average Score: 24.47
Episode 800      Average Score: 24.08
Episode 900      Average Score: 23.52
Episode 1000     Average Score: 23.95
Episode 1100     Average Score: 24.44
Episode 1200     Average Score: 24.19
Episode 1300     Average Score: 24.08
Episode 1400     Average Score: 23.75
Episode 1500     Average Score: 23.25
Episode 1600     Average Score: 24.23
Episode 1700     Average Score: 23.26
Episode 1800     Average Score: 24.01
Episode 1900     Average Score: 23.80
Episode 2000     Average Score: 23.46
```