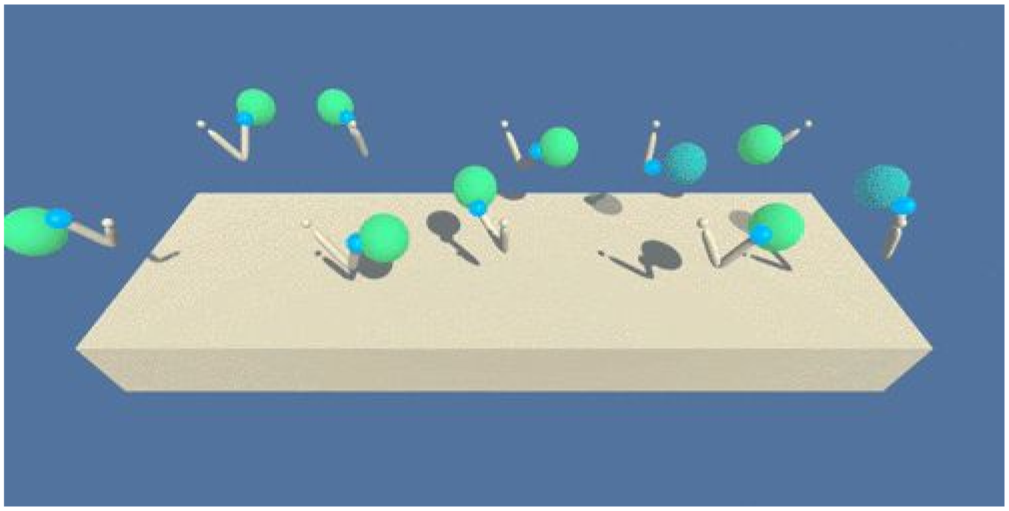


Project’s Goal

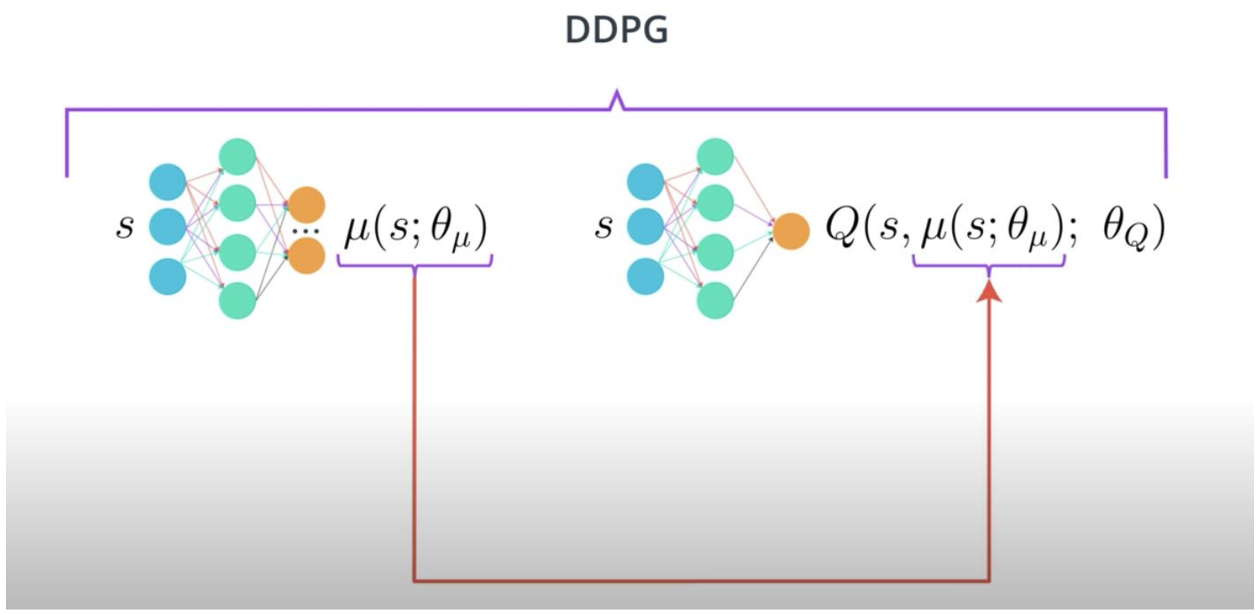
Information

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.

Learning Algorithm

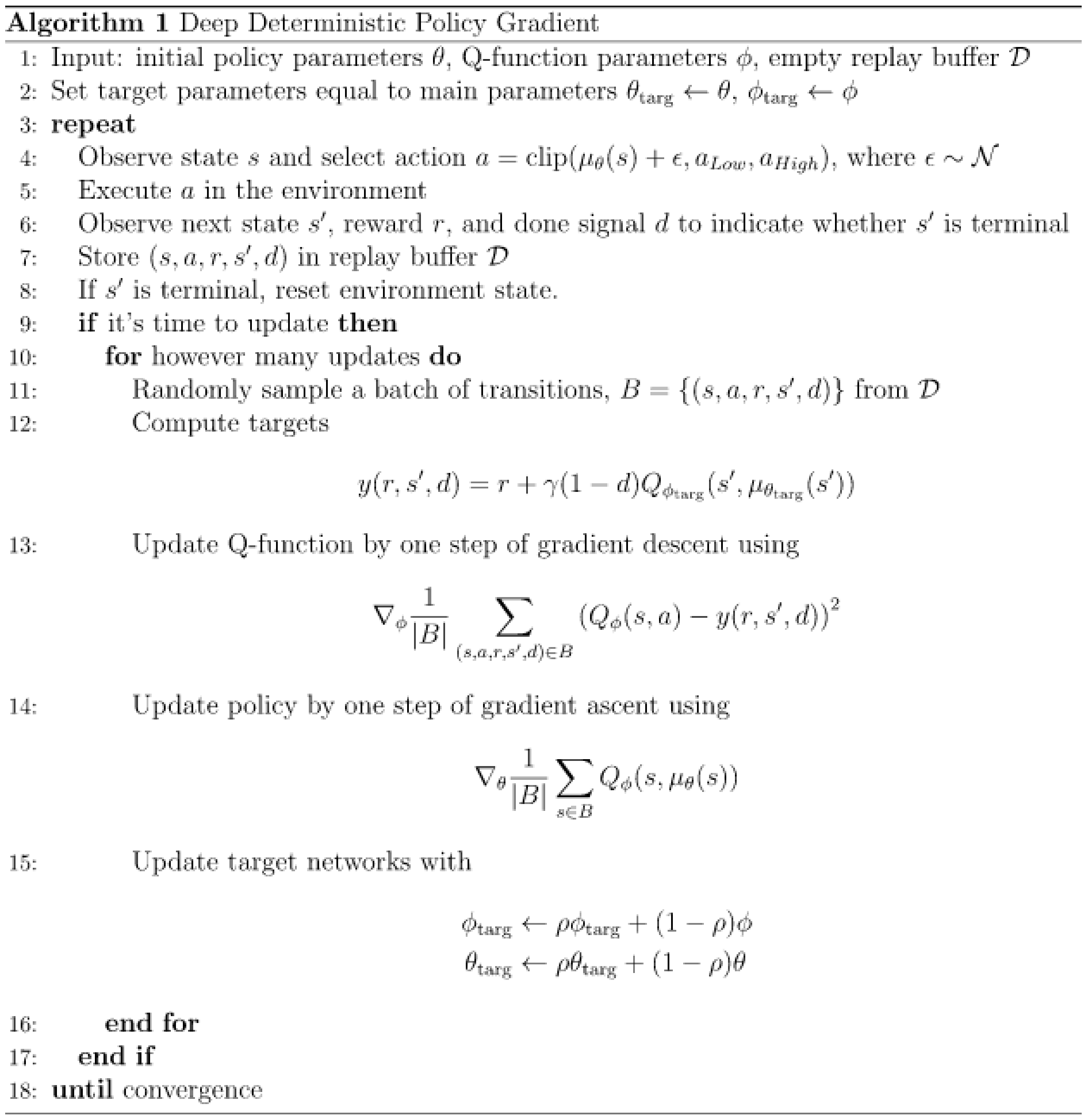
In this project, the agent is trained by the DDPG Algorithm.



DDPG algorithm;

- Actor-critic, model-free algorithm based on the deterministic policy gradient that can operate over continuous action spaces.
- is able to find policies whose performance is competitive with those found by a planning algorithm with full access to the dynamics of the domain and its derivatives.

Algorithm of DDPG;

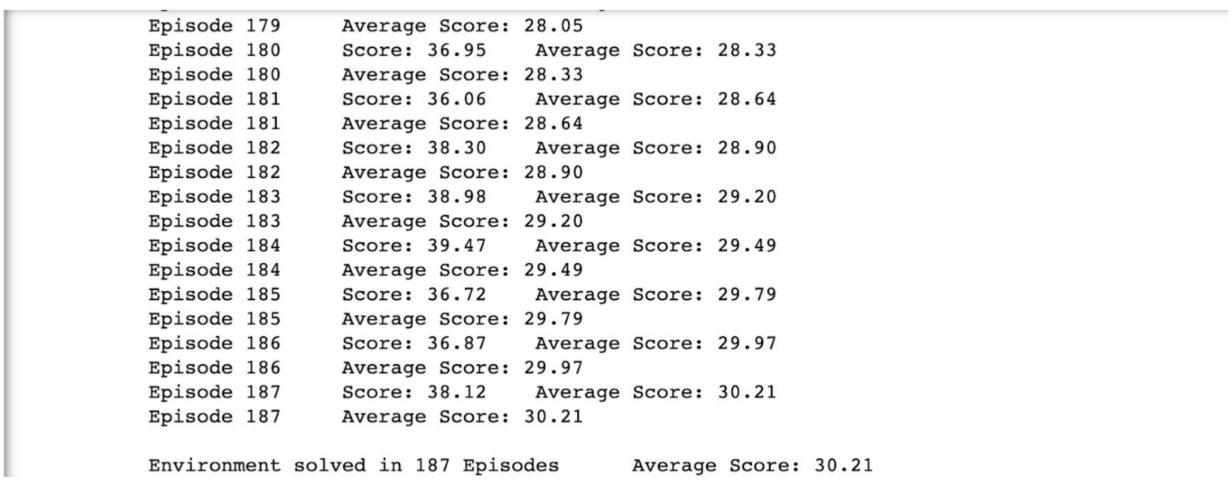


Algorithm 1 Deep Deterministic Policy Gradient

```
1: Input: initial policy parameters $\theta$, Q-function parameters $\phi$, empty replay buffer $\mathcal{D}$
2: Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
3: **repeat**
4:     Observe state $s$ and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
5:     Execute $a$ in the environment
6:     Observe next state $s'$, reward $r$, and done signal $d$ to indicate whether $s'$ is terminal
7:     Store $(s, a, r, s', d)$ in replay buffer $\mathcal{D}$
8:     If $s'$ is terminal, reset environment state.
9:     **if** it's time to update **then**
10:        **for** however many updates **do**
11:            Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
12:            Compute targets
                   $$y(r, s', d) = r + \gamma(1 - d) Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$
13:            Update Q-function by one step of gradient descent using
                   $$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_\phi(s, a) - y(r, s', d)\right)^2$$
14:            Update policy by one step of gradient ascent using
                   $$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$$
15:            Update target networks with
                   $$\phi_{\text{targ}} \leftarrow \rho \phi_{\text{targ}} + (1 - \rho)\phi$$
                   $$\theta_{\text{targ}} \leftarrow \rho \theta_{\text{targ}} + (1 - \rho)\theta$$
16:        **end for**
17:    **end if**
18: **until** convergence
```

Parameters and Results:

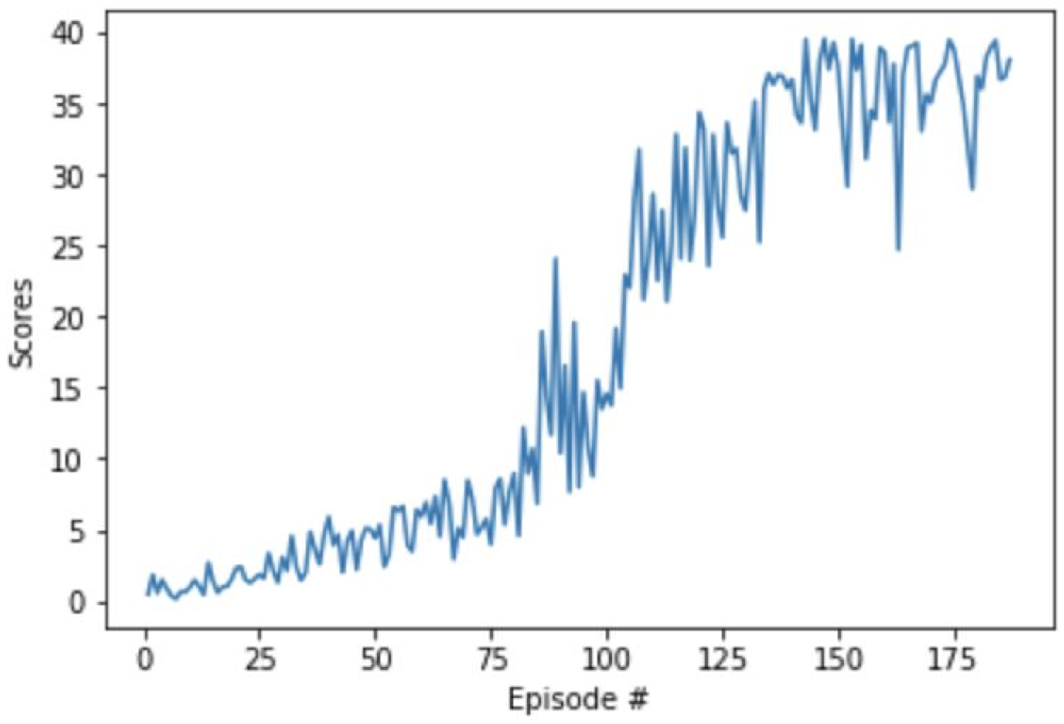
```
BUFFER_SIZE = int(1e5)  # replay buffer size
BATCH_SIZE = 128        # minibatch size
GAMMA = 0.99            # discount factor
TAU = 1e-3              # for soft update of target parameters
LR_ACTOR = 2e-4         # learning rate of the actor
LR_CRITIC = 2e-4        # learning rate of the critic
WEIGHT_DECAY = 0        # L2 weight decay
```



```
Episode 179     Average Score: 28.05
Episode 180     Score: 36.95    Average Score: 28.33
Episode 180     Average Score: 28.33
Episode 181     Score: 36.06    Average Score: 28.64
Episode 181     Average Score: 28.64
Episode 182     Score: 38.30    Average Score: 28.90
Episode 182     Average Score: 28.90
Episode 183     Score: 38.98    Average Score: 29.20
Episode 183     Average Score: 29.20
Episode 184     Score: 39.47    Average Score: 29.49
Episode 184     Average Score: 29.49
Episode 185     Score: 36.72    Average Score: 29.79
Episode 185     Average Score: 29.79
Episode 186     Score: 36.87    Average Score: 29.97
Episode 186     Average Score: 29.97
Episode 187     Score: 38.12    Average Score: 30.21
Episode 187     Average Score: 30.21

Environment solved in 187 Episodes      Average Score: 30.21
```

Plot of Rewards



Ideas for Future Work:

- **Proximal Policy Optimization**

Proximal Policy Optimization (PPO), which perform comparably or better than state-of-the-art approaches while being much simpler to implement and tune. Instead of imposing a hard constraint, it formalizes the constraint as a penalty in the objective function.

- **Prioritized Experience Replay**

It is built on top of experience replay buffers, which allow a reinforcement learning agent to store experiences in the form of transition tuples with states, actions, rewards, and successor states at some time index. In contrast to consuming samples online and discarding them thereafter, sampling from the stored experiences means they are less heavily “correlated” and can be re-used for learning

- **Asynchronous Advantage Actor-Critic (A3C)**

A3C consists of multiple independent agents(networks) with their own weights, who interact with a different copy of the environment in parallel. Thus, they can explore a bigger part of the state-action space in much less time