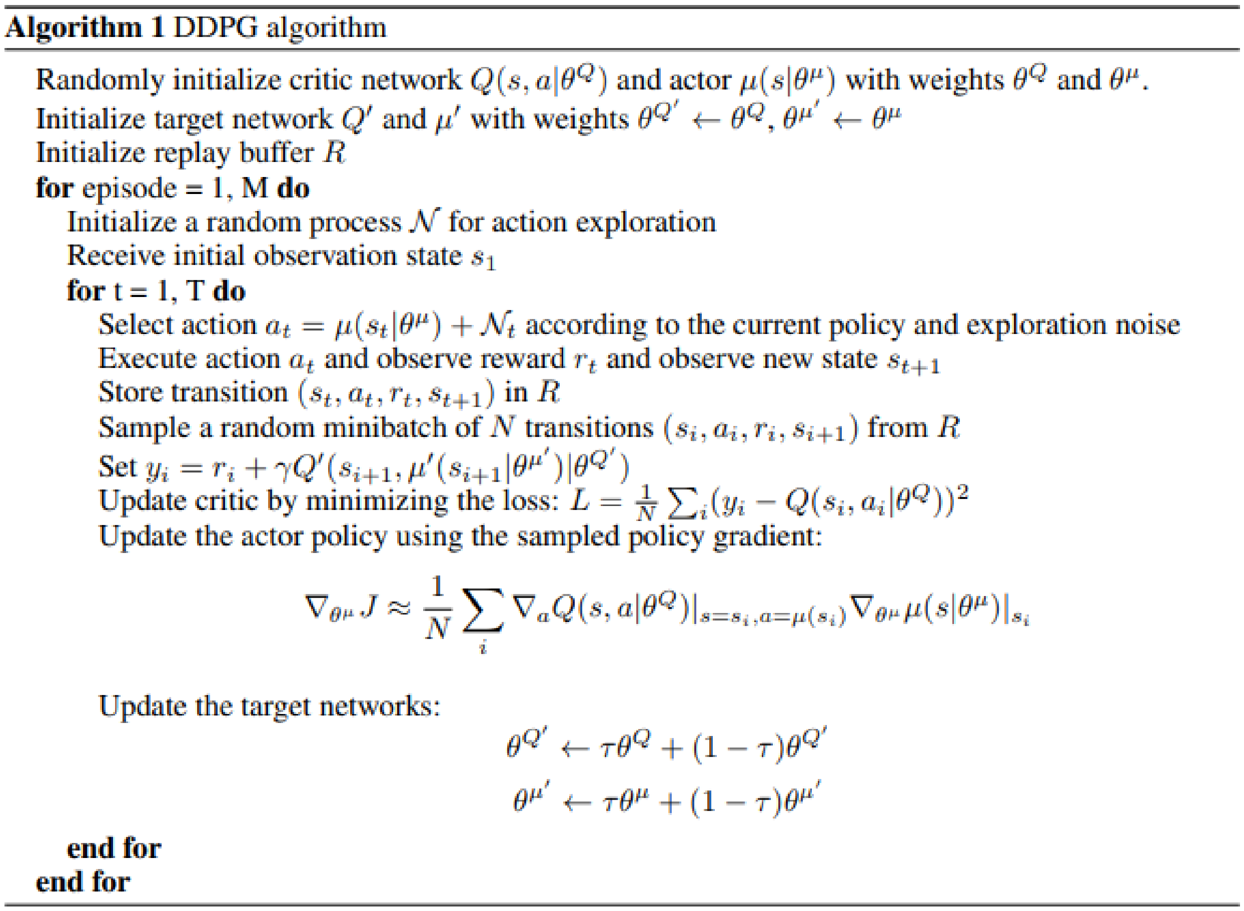
Report

Learning Algorithm

The Agent was designed using **Deep Deterministic Policy Gradients (DDPG) Algorithm.** The algorithm is better depicted in the following pseudocode.



```
Algorithm 1 DDPG algorithm
 Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
 Initialize target network $Q'$ and $\mu'$ 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
```

Figure 1 Deep Deterministic Policy Gradients (DDPG) Algorithm. Source: https://arxiv.org/pdf/1509.02971.pdf

Firstly, we initiate 2 critic networks and 2 actor networks. The two critic and actor networks consist of target and local networks. The local critic network can be written as $Q(s, a|\theta^Q)$ and the target as $Q'(s, a|\theta^{Q'})$. Like the critic networks, the target networks consist of the local $\mu(s, a|\theta^\mu)$ and the target $\mu'(s, a|\theta^{\mu'})$. The local and target networks are initialized with the same random weights applied both for the actor and critic. Then, we need to initialize the replay buffer *R* for the experience replay. The replay buffer *R* will store all the experiences that the agent interacts in the environment.

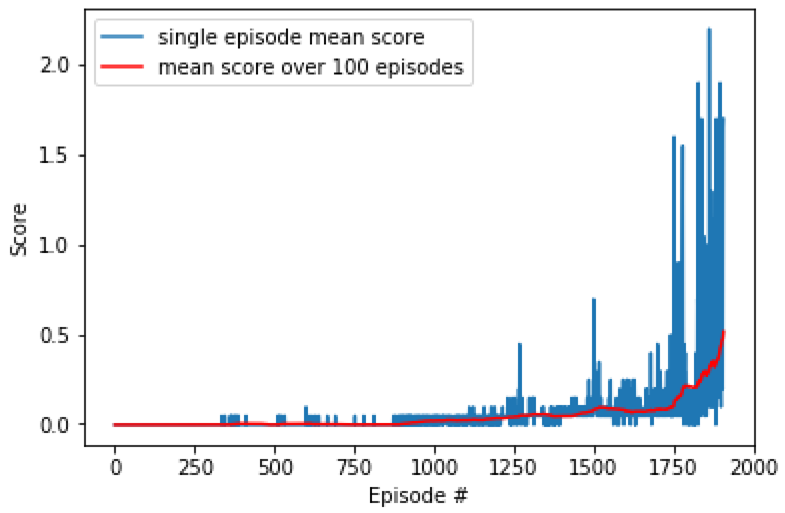
After all the initializations are complete, the agent is ready to interact with the environment through number of episodes. The number of episodes is one of the hyperparameters used in the training process. Within every episode, we need to initialize a random process *N* for the agent exploration and take the initial state s_1 from the environment. We begin the learning process with limited timesteps *T* which is another learning hyperparameter. In this case, the agent is only allowed to make maximum timestep *T*. Within the learning timestep, the agent begins with taking an action $a_t = \mu(s_t|\theta^\mu) + N_t$ from the local actor network that refers to current policy with additional exploration noise N_t. The agent then receives the reward r_t and new state s_{t+1} in effect of the action a_t taken. Those experience variables (s_t, a_t, r_t, s_{t+1}) then are saved in the replay buffer *R*. The agent begins the learning process by calculating the Q targets y_i and then compute the critic loss by calculating the MSE loss between Q targets y_i and the output of local critic network. Using the loss the agent takes the learning step to minimize the critic loss. The learning process is continued for the actor networks by calculating the Q value from the local critic network and use the calculation results as the actor loss. The actor then takes the learning step using the loss for the local network. After taking one learning step for both the actor and critic, we need to do soft updates for the actor and critic target networks with specified τ hyperparameter. The process is the continued until all the episodes *M* elapsed.

In this case, we used two agents played to gain the highest average score. Thus, we used two DDPG agents learn with their own experience and networks.

Training Process

The average score of 0.51 was reached at 1908 iterations as depicted in the chart below:

```
Episode 1700    Average Score: 0.08
Episode 1800    Average Score: 0.21
Episode 1900    Average Score: 0.43
Episode 1908    Average Score: 0.51
Environment solved in 1908 episodes!    Average Score: 0.51
```



Hyperparameters

Below is the table of all hyperparameters along with their training values and descriptions.

Hyperparameter	Value	Description
Replay Memory Size	1000000	The capacity of buffer *N*

Minibatch size	256	The length of experience sequence from the replay buffer used for training
Discount factor	0.99	Discount factor gamma used for Q-learning update
tau	0.001	soft update of target parameters
Learning rate for the actor	0.0002	The learning rate used for the optimizer for the actor
Learning rate for the critic	0.0002	The learning rate used for the optimizer for the critic
Weight Decay	0	L2 Weight Decay for the adam optimizer for the critic
Gradient Clipping	1	
max iterations	5000	The maximum iterations/episodes
Max timesteps	unlimited	The maximum timesteps
Actor Network Architecture	[256, 128]	Neural network architecture for the actor
Critic Network Architecture	[256, 128]	Neural network architecture for the critic

Future Improvement

Here are some ideas of future improvement:

1. **Multi-Agent Learning** with **sharing networks** and **experience replay** for faster learning and comprehensive states.
2. **Trust Region Policy Optimization (TRPO) and Truncated Natural Policy Gradient (TNPG)** for more stable learning which can result better performance.