1. DESCRIPTION OF THE IMPLEMENTATION:

The problem solved in this project is the Reacher problem as the second project of the Deep Reinforcement learning nanodegree. We chose the second environment with 20 agents to have a faster training process. The algorithm applied to solve the project is the DDPG (Deep Deterministic Policy Gradient) figure 1.

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

Figure 1 *DDPG Algorithm*

2. REACHER SOLUTION

The solution of the Reacher problem is contained in the jupyter notebook named “Continuous_Control.ipynb” This notebook uses two files the first one contains the code for the deep ddpg agent and the name of this file is “ddpg_agent.py”. The code inside this class also uses another file called “model.py” and it uses deep neural network with to define the actors and critics for DDPG algorithm. The architecture of the neural networks are shown below.

Actor Deep Neural Network				
	Input Size	**Output Size**	**Layer Type**	**Activation Function**
Layer 1	33	128	nn.Linear	Relu
Layer 2	150	128	nn.BatchNorm1d	-
Layer 3	150	128	nn.Linear	Relu
Layer 3	150	4	nn.Linear	Tanh

Table 1 *Actor and Actor target dnn architecture*

Critic Deep Neural Network				
	Input Size	**Output Size**	**Layer Type**	**Activation Function**
Layer 1	33	128	nn.Linear	Relu
Layer 2	128	128	nn.BatchNorm1d	-
Layer 3	132	128	nn.Linear	Relu
Layer 3	128	1	nn.Linear	-

__Table 2__ Critic and Critic target dnn architecture

The hyperparameters are.

Hyperparameters	
Replay buffer size	1e5
Minibatch size	128
GAMMA (discount factor)	0.99
TAU (soft update)	1e-3
Learning rate	2e-4
Weight decay	0

__Table 3__ Hyperparameters

The number of episodes used for training were 245 episodes and after running the training we observe the following graph that shows how the average reward increases over time. One important aspect is that the average reward at episode 50 is almost 30. This tell us that using multiple agents makes the training fast

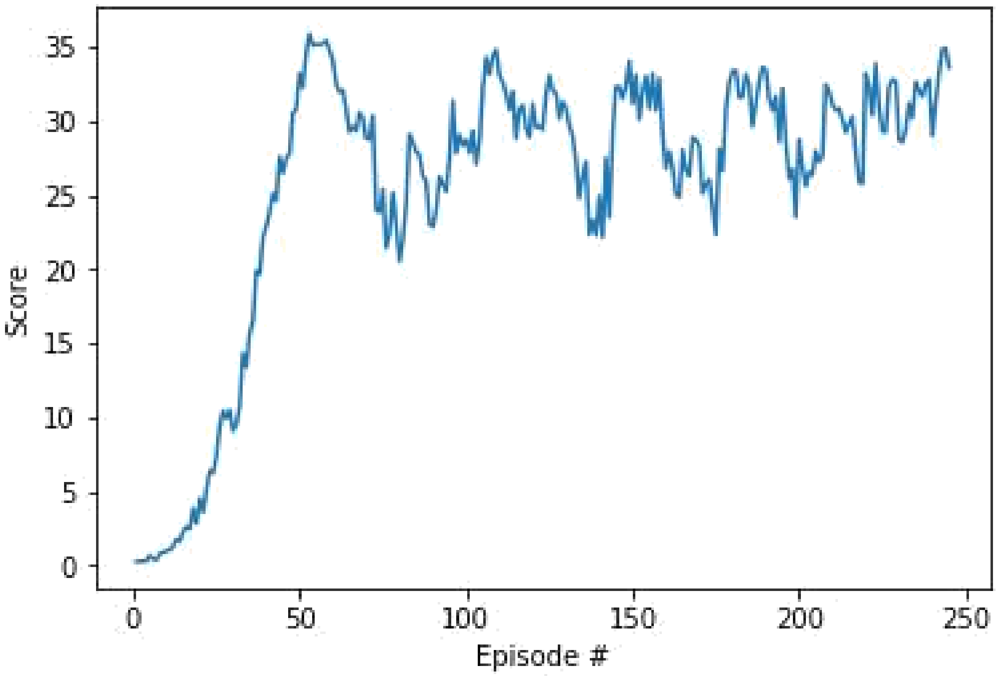

__Table 2__ reward plot

The weights for the trained model are stored in two files named “checkpoint_actor.pth” and “checkpoint_critic.pth” that corresponds to the weights of the actor and the critic model.

3. FUTURE WORK

To improve the performance and the training speed of DDPG algorithm GAE (Generalized Advantage Estimation) can be added. There are other algorithms that can be implemented to solve this environment like Proximal Policy Optimization or A2C.