Report

The solution is based on DDPG (Deep Deterministic Policy Gradient) architecture.

The foundation for the solution was provided in the Udacity repository for the Deep Reinforcement Learning Nanodegree (https://github.com/udacity/deep-reinforcement-learning)

While the training approach is based on the agent.py in the repository folder ddpg-pendulum, the model.py file is based on the model definition provided in the ddpg-bipedal folder of the same repository. Agent.py has been modified to accommodate for multiple agents in the environment, while model.py has not been changed from its original version (except of the fc_unit settings).

1. **Environment Description**

```
INFO:unityagents:
'Academy' started successfully!
Unity Academy name: Academy
        Number of Brains: 1
        Number of External Brains : 1
        Lesson number : 0
        Reset Parameters :
                goal_speed -> 1.0
                goal_size -> 5.0
Unity brain name: ReacherBrain
        Number of Visual Observations (per agent): 0
        Vector Observation space type: continuous
        Vector Observation space size (per agent): 33
        Number of stacked Vector Observation: 1
        Vector Action space type: continuous
        Vector Action space size (per agent): 4
        Vector Action descriptions: , , ,
```

2. **State and action spaces**

```
Number of agents: 20
Size of each action: 4
There are 20 agents. Each observes a state with length: 33
The state for the first agent looks like: [  0.00000000e+00  -
4.00000000e+00   0.00000000e+00   1.00000000e+00
  -0.00000000e+00  -0.00000000e+00  -4.37113883e-08   0.00000000e+00
   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00   0.00000000e+00  -1.00000000e+01   0.00000000e+00
   1.00000000e+00  -0.00000000e+00  -0.00000000e+00  -4.37113883e-08
   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00   0.00000000e+00   5.75471878e+00  -1.00000000e+00
   5.55726624e+00   0.00000000e+00   1.00000000e+00   0.00000000e+00
  -1.68164849e-01]
```

3. **Model Description (actor_target, critic_target):**

```
Actor(
  (fc1): Linear(in_features=33, out_features=256, bias=True)
  (fc2): Linear(in_features=256, out_features=4, bias=True)
```

```
)
Critic(
  (fcs1): Linear(in_features=33, out_features=256, bias=True)
  (fc2): Linear(in_features=260, out_features=256, bias=True)
  (fc3): Linear(in_features=256, out_features=128, bias=True)
  (fc4): Linear(in_features=128, out_features=1, bias=True)
)
```

The summary above provides the description of the underlying deep neural networks used in the approach as part of this solution. The solution uses two networks, one for actor and one for the critic.

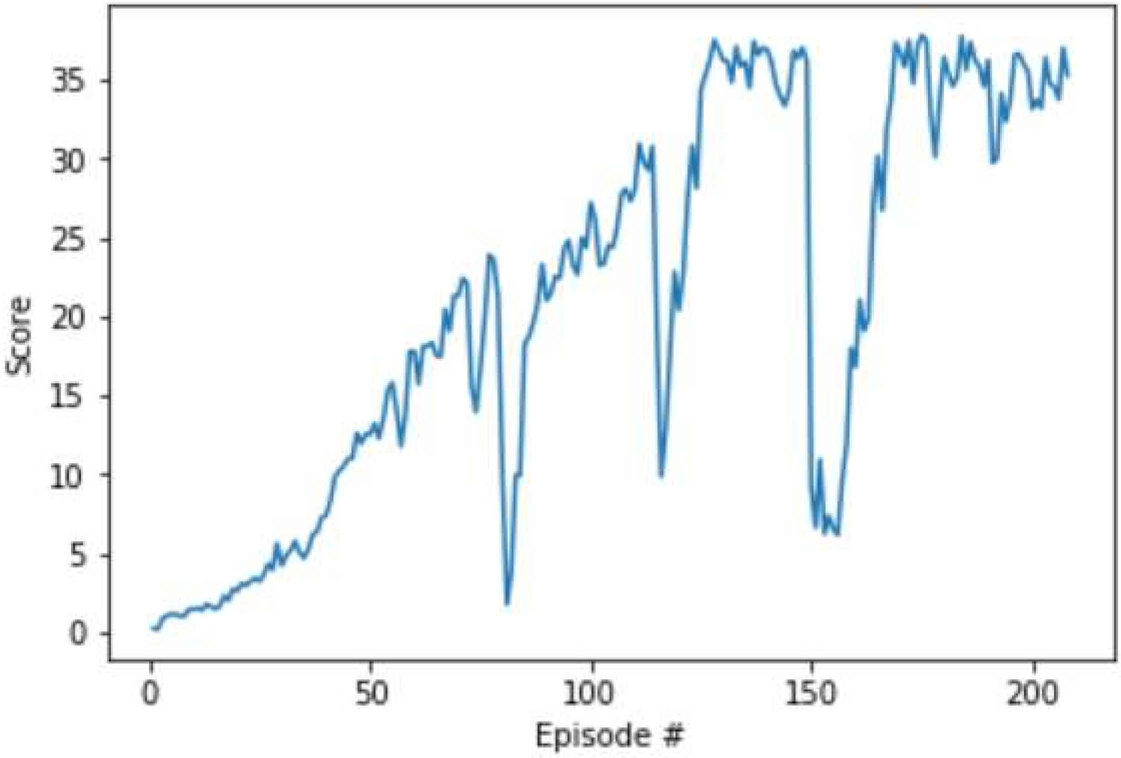
4. Hyperparameter:

Parameter	Value
n_episodes (int): maximum number of training episodes	**2000**
max_t (int): maximum number of timesteps per episode	**1000**
num_agents : amount of agents in the environment	**20**
DDPG Agent HyperParameters	
BUFFER_SIZE: replay buffer size	**int(1e5)**
BATCH_SIZE: minibatch size	**64**
GAMMA: discount factor	**0.99**
TAU: for soft update of target parameters	**1e-3**
LR_ACTOR: learning rate for actor	**1e-4**
LR_CRITIC: learning rate for critic	**1e-4**
WEIGHT_DECAY: L2 weigth decay	**0**

5. Training Summary:

```
....
Episode 202    Mean score (last 100 episodes): 29.49
Episode 203    Mean score (last 100 episodes): 29.62
Episode 204    Mean score (last 100 episodes): 29.72
Episode 205    Mean score (last 100 episodes): 29.82
Episode 206    Mean score (last 100 episodes): 29.91
Episode 207    Mean score (last 100 episodes): 30.00
Episode 208    Mean score (last 100 episodes): 30.07

Environment solved in 208 episodes! Mean score (last 100 episodes): 30.07
```



The plot above shows the agents score achieved during the training episodes. the agent can receive an average reward (over 100 episodes, and over all 20 agents) of >30 after 208 episodes.

6. **Ideas for optimizing the solution:**

- **Optimizing hyperparamters:** During the development of the exercise I experimented with the hyperparameters (especially with the fc_units, learning rates, tau and batch size) I strongly believe that there is plenty of room of further optimization here with automating the experimentation efforts. Creating a script that changes parameters based on statistical methods and combining it with an automated AWS script to spin up multiple environments, can give a good solution to scale experimentation efforts.
- **Changing the Algorithm:** As an option Proximal Policy Optimization (PPO) and Distributed Distributional Deterministic Policy Gradients (D4PG) methods could be explored.