1. Environment

We use the option 1 of the two environment. The environment contains a single agent. We train a 2 DOF robotic arm to reach a particular target in the field. The reward is the sum of reward_dist and reward_ctrl. reward_dist is the distance between the fingertip of the robot and the target. reward_ctrl is the sum of squared actions. The state space is 33 dimensions. The action space is 2 dimensions and the values can be between -1 and 1. The action controls the joints of the robot. The problem is considered solved if the average reward is at least 30+ over 100 episodes.

2. Algorithm

We use Deep Deterministic Policy Gradient (DDPG) to implement reinforcement learning. DDPG was more suitable than DQN because the environment of the reacher had continuous action space. DDPG has two deep neural networks: a actor network and a critic network. The actor is used to determine the optical policy deterministically. The critic evaluates the best belief action from the actor. For each network, there is a real network and a target network. The real network and the target network is used to increase stability. The real network is actively trained and soft update is applied to the target network, slowly blending the network weights. For training, the DDPG uses a replay buffer to store experiences.

3. Hyper parameters
The following are the hyper parameters

In ddpg_agent.py,

```
BUFFER_SIZE = int(1e5)      # replay buffer size
BATCH_SIZE = 200            # minibatch size
GAMMA = 0.90                # discount factor
TAU = 1e-3                  # for soft update of target parameters
LR_ACTOR = 1e-4             # learning rate of the actor
LR_CRITIC = 1e-4            # learning rate of the critic
WEIGHT_DECAY = 0            # L2 weight decay
```

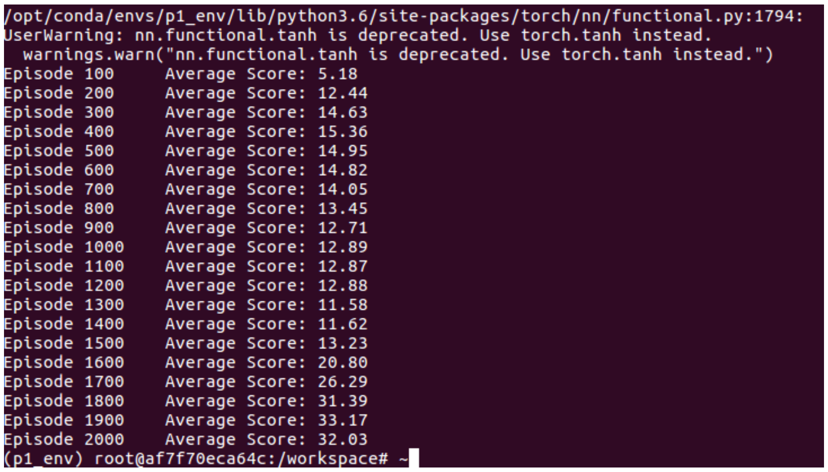
In main.py,
```
Train = 1                   # whether to train or test
train_n_episode = 2000      # number of episode for training
test_n_episode = 100        # number of episode for testing
```

4. Architecture
Deep Deterministic Policy Gradient (DDPG) consists of an actor network and a critic network. The actor maps the state to action. The actor network has two hidden layers. The size of the hidden layers are 512 and 128. The critic maps the state and action pairs to q-values. The critic network has

three hidden layers. The size of the hidden layers are 512, 128 and 64. For stability, each network has a real and target network.

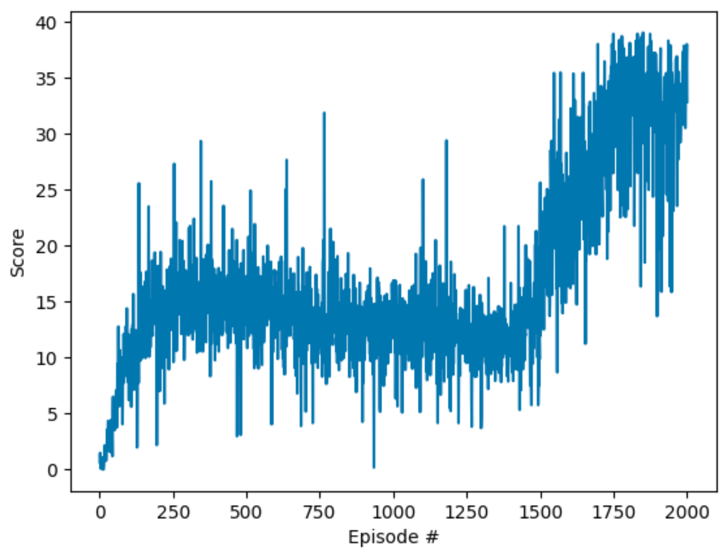
5. Results & Figures

The DDPG agent was trained for 2000 episodes. After 1800 episodes, the average score was greater than 30. We observed that during training, the score fluctuates a lot. The average scores for episode 500 to 1000 seemed greater than the average scores for episode 1000 to 1500. After training for 1800 episodes, the scores became above 30. The average score printed on the terminal is shown below.

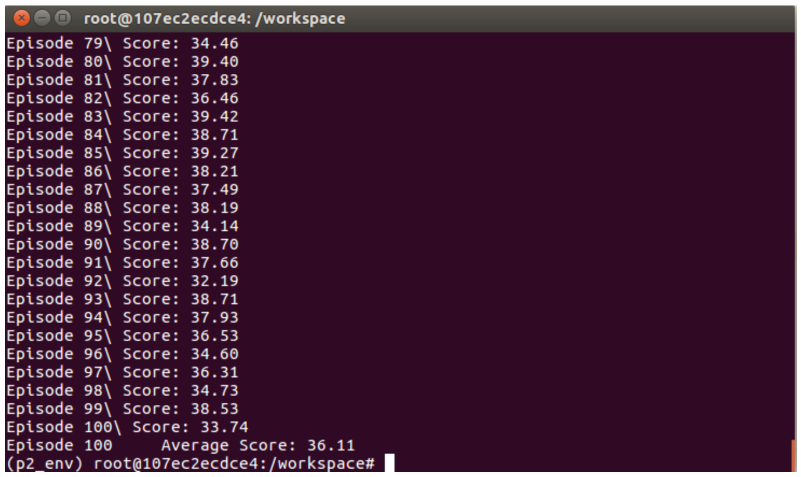


```
/opt/conda/envs/p1_env/lib/python3.6/site-packages/torch/nn/functional.py:1794:
UserWarning: nn.functional.tanh is deprecated. Use torch.tanh instead.
  warnings.warn("nn.functional.tanh is deprecated. Use torch.tanh instead.")
Episode 100     Average Score: 5.18
Episode 200     Average Score: 12.44
Episode 300     Average Score: 14.63
Episode 400     Average Score: 15.36
Episode 500     Average Score: 14.95
Episode 600     Average Score: 14.82
Episode 700     Average Score: 14.05
Episode 800     Average Score: 13.45
Episode 900     Average Score: 12.71
Episode 1000    Average Score: 12.89
Episode 1100    Average Score: 12.87
Episode 1200    Average Score: 12.88
Episode 1300    Average Score: 11.58
Episode 1400    Average Score: 11.62
Episode 1500    Average Score: 13.23
Episode 1600    Average Score: 20.80
Episode 1700    Average Score: 26.29
Episode 1800    Average Score: 31.39
Episode 1900    Average Score: 33.17
Episode 2000    Average Score: 32.03
(p1_env) root@af7f70eca64c:/workspace# ~
```

We also plotted the scores during training below. The score has an increasing trend in 0 to 250 episodes. From 500 to 1000 episodes, it seems to have a downward trend. Starting episode 1500, the score of the agent starts to increase significantly.

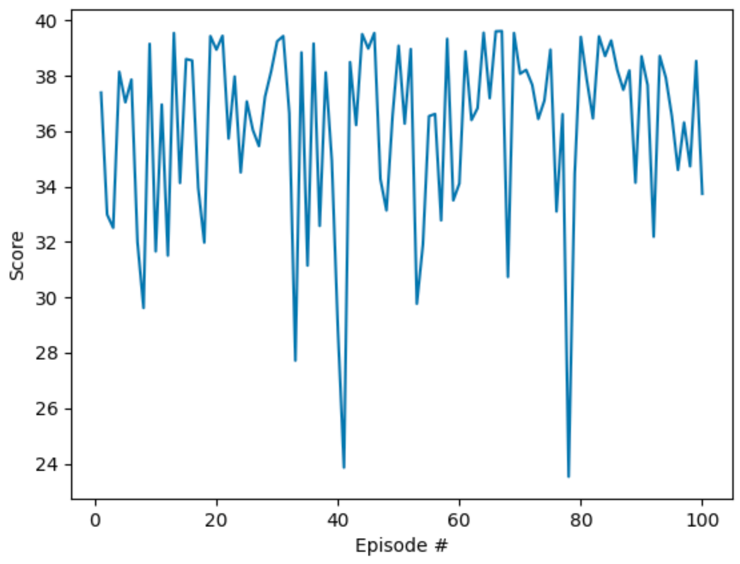


When we average the 100 episode while testing, the average score of the agent was 36.11 like shown below. We can see that after training the agent, the agent is able to score above 30 in average.



```
Episode 79\ Score: 34.46
Episode 80\ Score: 39.40
Episode 81\ Score: 37.83
Episode 82\ Score: 36.46
Episode 83\ Score: 39.42
Episode 84\ Score: 38.71
Episode 85\ Score: 39.27
Episode 86\ Score: 38.21
Episode 87\ Score: 37.49
Episode 88\ Score: 38.19
Episode 89\ Score: 34.14
Episode 90\ Score: 38.70
Episode 91\ Score: 37.66
Episode 92\ Score: 32.19
Episode 93\ Score: 38.71
Episode 94\ Score: 37.93
Episode 95\ Score: 36.53
Episode 96\ Score: 34.60
Episode 97\ Score: 36.31
Episode 98\ Score: 34.73
Episode 99\ Score: 38.53
Episode 100\ Score: 33.74
Episode 100     Average Score: 36.11
(p2_env) root@107ec2ecdce4:/workspace#
```

We also plotted the scores during testing below. There are outliers, but in general the scores are above 30.



6. Future Idea

Training with a higher number of episodes - In future work, we should check if performance increases as the number of training episodes increase. When we trained the ddpg, we saw a sudden jump in average score. There could be other significant increase in average score if we trained for a longer period of time.

Configure the hyper parameters - In future work, more testing should be done with the hyper parameters. Some of the hyper parameters directly influence the agent’s learning behavior. Each hyper parameter should be tested in the future and configured more to increase performance.

Try a different algorithm - In future work, we can try implementing PPO.

Attempt the optional challenge - In this work, we used a single agent to train and test the ddpg algorithm. There was another option (option 2) to use 20 agents to train the networks. When training 20 agents, it should be more efficient if we can train the agents simultaneously using threading. Training multiple agents simultaneously needs to be explored.