Deep Reinforcement Learning Nanodegree Project 2 Report

Ohn Kim

Learning Algorithm

I used DDPG algorithm to solve this problem.

Actor-Critic is a mix of policy-based and value-based methods. Policy based agent (actor) determines what action to take, and value-based agent (critical) determines the value of the current state and action. The representative algorithm of Actor-critical learning, DDPG (Deep Deterministic Policy Gradients), is learned in the following ways:

1. Actor makes an action based on state.
2. Critical is taught to predict the reward based on state, action and to create a value like the actual reward.
3. The actor receives an expected reward by delivering the action he created from state to critical and learns to maximize the expected reward.

Hyperparameters

- BUFFER_SIZE = int(1e6) # replay buffer size
- BATCH_SIZE = 1024 # minibatch size
- GAMMA = 0.99 # discount factor
- TAU = 1e-3 # for soft update of target parameters
- Actor LR = 1e-4 # actor learning rate
- Critic LR = 3e-4 # critic learning rate
- n_episodes = 500 # maximum number of training episodes
- max_t = 1000 # maximum number of time steps per episode
- leak = 0.01 # leakyReLU

Model architecture

The structure of the actor and critical model is as follows. The fully connected layer is connected by leaky_ReLU, and in the case of the actor, tanh activation function was applied at the end.

Actor

```
state = self.bn(state)
x = F.leaky_relu(self.fc1(state), negative_slope=self.leak)
x = F.leaky_relu(self.fc2(x), negative_slope=self.leak)
x =  torch.tanh(self.fc3(x))

fc1=256, fc2=128.
```

Critic

```
state = self.bn(state)
x = F.leaky_relu(self.fcs1(state), negative_slope=self.leak)
x = torch.cat((x, action), dim=1)
x = F.leaky_relu(self.fc2(x), negative_slope=self.leak)
x = F.leaky_relu(self.fc3(x), negative_slope=self.leak)
x =  self.fc4(x)
```

fc1=256, fc2=128, fc3=128

Batch normalization and leaky_ReLU function are applied.

```
# size of each action
action_size = brain.vector_action_space_size
print('Size of each action:', action_size)

# examine the state space
states = env_info.vector_observations
state_size = states.shape[1]
print('There are {} agents. Each observes a state with length: {}'.format(states.shape[0], state_size))
```

```
Size of each action: 4
There are 20 agents. Each observes a state with length: 33
```

Action_size = 4, state_size = 33

of episodes needed to solve the environment

```
Episode: 10      Average Score: 0.29     Current Score: 0.04
Episode: 20      Average Score: 0.52     Current Score: 0.98
Episode: 30      Average Score: 0.81     Current Score: 1.89
Episode: 40      Average Score: 1.23     Current Score: 2.84
Episode: 50      Average Score: 1.81     Current Score: 4.37
Episode: 60      Average Score: 2.53     Current Score: 7.54
Episode: 70      Average Score: 3.26     Current Score: 7.73
Episode: 80      Average Score: 4.13     Current Score: 10.50
Episode: 90      Average Score: 4.93     Current Score: 12.78
Episode: 100     Average Score: 5.74     Current Score: 14.21
Episode: 110     Average Score: 7.12     Current Score: 13.75
Episode: 120     Average Score: 8.47     Current Score: 15.36
Episode: 130     Average Score: 9.81     Current Score: 14.94
Episode: 140     Average Score: 11.18    Current Score: 18.04
Episode: 150     Average Score: 12.64    Current Score: 20.53
Episode: 160     Average Score: 14.18    Current Score: 21.73
Episode: 170     Average Score: 15.74    Current Score: 22.94
Episode: 180     Average Score: 17.14    Current Score: 24.58
Episode: 190     Average Score: 18.54    Current Score: 25.91
Episode: 200     Average Score: 19.79    Current Score: 27.81
Episode: 210     Average Score: 21.07    Current Score: 25.75
Episode: 220     Average Score: 22.31    Current Score: 27.28
Episode: 230     Average Score: 23.59    Current Score: 29.44
Episode: 240     Average Score: 24.83    Current Score: 28.54
Episode: 250     Average Score: 25.94    Current Score: 29.71
Episode: 260     Average Score: 26.80    Current Score: 31.08
Episode: 270     Average Score: 27.53    Current Score: 28.77
Episode: 280     Average Score: 28.22    Current Score: 31.18
Episode: 290     Average Score: 28.72    Current Score: 28.48
Episode: 300     Average Score: 29.23    Current Score: 32.83
Episode: 310     Average Score: 29.64    Current Score: 31.67
Episode: 318     Average Score: 30.07    Current Score: 34.09
Environment solved in 218 episodes!    Average Score: 30.07
```

After a total of 218 episodes, average score is over +30.

Plot of rewards
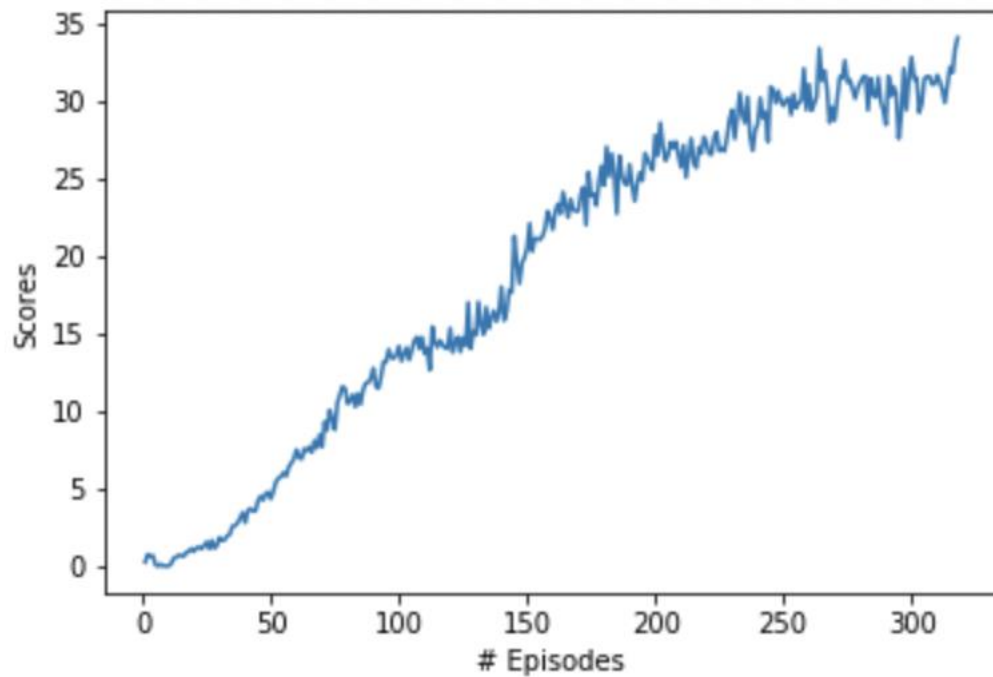
Ideas for Future Work

I used the DDPG model to solve this problem. This model could have solved the problem enough, but other algorithms could be used for better performance. Using algorithms (ex. A3C, A2C, PPO) could achieve better results. And more optimization of hyperparameters is needed.